Red Hat Enterprise Linux 6 Security-Enhanced Linux

User Guide
Edition 6

Barbora Ančincová

Red Hat Enterprise Linux 6 Security-Enhanced Linux

User Guide
Edition 6

Barbora Ančincová
Red Hat Customer Content Services
bancinco@redhat.com

Legal Notice

Abstract

This guide assists users and administrators in managing and using Security-Enhanced Linux.

Table of Contents

Preface

The Red Hat Enterprise Linux 6 SELinux User Guide is for people with minimal or no experience with SELinux. Although system administration experience is not necessary, content in this guide is written for system administration tasks. This guide provides an introduction to fundamental concepts and practical applications of SELinux. After reading this guide you should have an intermediate understanding of SELinux.

Thank you to everyone who offered encouragement, help, and testing - it is most appreciated. Very special thanks to:

- Dominick Grift, Stephen Smalley, and Russell Coker for their contributions, help, and patience.

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keys and key combinations. For example:

> To see the contents of the file **`my_next_bestselling_novel`** in your current working directory, enter the **`cat my_next_bestselling_novel`** command at the shell prompt and press **`Enter`** to execute the command.

The above includes a file name, a shell command and a key, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from an individual key by the plus sign that connects each part of a key combination. For example:

> Press **`Enter`** to execute the command.
>
> Press **`Ctrl`**+**`Alt`**+**`F2`** to switch to a virtual terminal.

The first example highlights a particular key to press. The second example highlights a key combination: a set of three keys pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **`mono-spaced bold`**. For example:

> File-related classes include **`filesystem`** for file systems, **`file`** for files, and **`dir`** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog-box text; labeled buttons; check-box and radio-button labels; menu titles and submenu titles. For example:

Choose **System → Preferences → Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, select the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).

To insert a special character into a **gedit** file, choose **Applications → Accessories → Character Map** from the main menu bar. Next, choose **Search → Find...** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit → Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

Mono-spaced Bold Italic or ***Proportional Bold Italic***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

To connect to a remote machine using ssh, type **ssh *username@domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh john@example.com**.

The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount /home**.

To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above: username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

Publican is a *DocBook* publishing system.

1.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **mono-spaced roman** and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in **mono-spaced roman** but add syntax highlighting as follows:

```
static int kvm_vm_ioctl_deassign_device(struct kvm *kvm,
                 struct kvm_assigned_pci_dev *assigned_dev)
{
        int r = 0;
```

```
        struct kvm_assigned_dev_kernel *match;

        mutex_lock(&kvm->lock);

        match = kvm_find_assigned_dev(&kvm->arch.assigned_dev_head,
                                      assigned_dev->assigned_dev_id);
        if (!match) {
                printk(KERN_INFO "%s: device hasn't been assigned
before, "
                  "so cannot be deassigned\n", __func__);
                r = -EINVAL;
                goto out;
        }

        kvm_deassign_device(kvm, match);

        kvm_free_assigned_device(kvm, match);

out:
        mutex_unlock(&kvm->lock);
        return r;
}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled “Important” will not cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla: http://bugzilla.redhat.com/ against the product **Red Hat Enterprise Linux.**

When submitting a bug report, be sure to mention the manual's identifier: doc-*SELinux_User_Guide* and version number: **6**.

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Chapter 1. Trademark Information

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

UNIX is a registered trademark of The Open Group.

Type Enforcement is a trademark of Secure Computing, LLC, a wholly owned subsidiary of McAfee, Inc., registered in the U.S. and in other countries. Neither McAfee nor Secure Computing, LLC, has consented to the use or reference to this trademark by the author outside of this guide.

Apache is a trademark of The Apache Software Foundation.

MySQL is a trademark or registered trademark of MySQL AB in the U.S. and other countries.

Other products mentioned may be trademarks of their respective corporations.

Chapter 2. Introduction

Security-Enhanced Linux (SELinux) is an implementation of a *mandatory access control* mechanism in the Linux kernel, checking for allowed operations after standard *discretionary access controls* are checked. It was created by the National Security Agency and can enforce rules on files and processes in a Linux system, and on their actions, based on defined policies.

When using SELinux, files, including directories and devices, are referred to as objects. Processes, such as a user running a command or the Mozilla Firefox application, are referred to as subjects. Most operating systems use a Discretionary Access Control (DAC) system that controls how subjects interact with objects, and how subjects interact with each other. On operating systems using DAC, users control the permissions of files (objects) that they own. For example, on Linux operating systems, users could make their home directories world-readable, giving users and processes (subjects) access to potentially sensitive information, with no further protection over this unwanted action.

Relying on DAC mechanisms alone is fundamentally inadequate for strong system security. DAC access decisions are only based on user identity and ownership, ignoring other security-relevant information such as the role of the user, the function and trustworthiness of the program, and the sensitivity and integrity of the data. Each user typically has complete discretion over their files, making it difficult to enforce a system-wide security policy. Furthermore, every program run by a user inherits all of the permissions granted to the user and is free to change access to the user's files, so minimal protection is provided against malicious software. Many system services and privileged programs run with coarse-grained privileges that far exceed their requirements, so that a flaw in any one of these programs could be exploited to obtain further system access. [1]

The following is an example of permissions used on Linux operating systems that do not run Security-Enhanced Linux (SELinux). The permissions and output in these examples may differ slightly from your system. Use the **`ls -l`** command to view file permissions:

```
~]$ ls -l file1
-rwxrw-r-- 1 user1 group1 0 2009-08-30 11:03 file1
```

In this example, the first three permission bits, **`rwx`**, control the access the Linux **`user1`** user (in this case, the owner) has to **`file1`**. The next three permission bits, **`rw-`**, control the access the Linux **`group1`** group has to **`file1`**. The last three permission bits, **`r--`**, control the access everyone else has to **`file1`**, which includes all users and processes.

Security-Enhanced Linux (SELinux) adds Mandatory Access Control (MAC) to the Linux kernel, and is enabled by default in Red Hat Enterprise Linux. A general purpose MAC architecture needs the ability to enforce an administratively-set security policy over all processes and files in the system, basing decisions on labels containing a variety of security-relevant information. When properly implemented, it enables a system to adequately defend itself and offers critical support for application security by protecting against the tampering with, and bypassing of, secured applications. MAC provides strong separation of applications that permits the safe execution of untrustworthy applications. Its ability to limit the privileges associated with executing processes limits the scope of potential damage that can result from the exploitation of vulnerabilities in applications and system services. MAC enables information to be protected from legitimate users with limited authorization as well as from authorized users who have unwittingly executed malicious applications. [2]

The following is an example of the labels containing security-relevant information that are used on processes, Linux users, and files, on Linux operating systems that run SELinux. This information is called the SELinux *context*, and is viewed using the **`ls -Z`** command:

```
~]$ ls -Z file1
-rwxrw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0      file1
```

In this example, SELinux provides a user (**unconfined_u**), a role (**object_r**), a type (**user_home_t**), and a level (**s0**). This information is used to make access control decisions. With DAC, access is controlled based only on Linux user and group IDs. It is important to remember that SELinux policy rules are checked *after* DAC rules. SELinux policy rules are not used if DAC rules deny access first.

Linux and SELinux Users

On Linux operating systems that run SELinux, there are Linux users as well as SELinux users. SELinux users are part of SELinux policy. Linux users are mapped to SELinux users. To avoid confusion, this guide uses “Linux user” and “SELinux user” to differentiate between the two.

2.1. Benefits of running SELinux

- All processes and files are labeled with a type. A type defines a domain for processes, and a type for files. Processes are separated from each other by running in their own domains, and SELinux policy rules define how processes interact with files, as well as how processes interact with each other. Access is only allowed if an SELinux policy rule exists that specifically allows it.
- Fine-grained access control. Stepping beyond traditional UNIX permissions that are controlled at user discretion and based on Linux user and group IDs, SELinux access decisions are based on all available information, such as an SELinux user, role, type, and, optionally, a level.
- SELinux policy is administratively-defined, enforced system-wide, and is not set at user discretion.
- Reduced vulnerability to privilege escalation attacks. One example: since processes run in domains, and are therefore separated from each other, and because SELinux policy rules define how processes access files and other processes, if a process is compromised, the attacker only has access to the normal functions of that process, and to files the process has been configured to have access to. For example, if the Apache HTTP Server is compromised, an attacker cannot use that process to read files in user home directories, unless a specific SELinux policy rule was added or configured to allow such access.
- SELinux can be used to enforce data confidentiality and integrity, as well as protecting processes from untrusted inputs.

However, SELinux is not:

- antivirus software,
- a replacement for passwords, firewalls, or other security systems,
- an all-in-one security solution.

SELinux is designed to enhance existing security solutions, not replace them. Even when running SELinux, it is important to continue to follow good security practices, such as keeping software up-to-date, using hard-to-guess passwords, firewalls, and so on.

2.2. Examples

The following examples demonstrate how SELinux increases security:

- The default action is deny. If an SELinux policy rule does not exist to allow access, such as for a process opening a file, access is denied.
- SELinux can confine Linux users. A number of confined SELinux users exist in SELinux policy. Linux users can be mapped to confined SELinux users to take advantage of the security rules and mechanisms applied to them. For example, mapping a Linux user to the SELinux **user_u** user, results in a Linux user that is not able to run (unless configured otherwise) set user ID (setuid) applications, such as **sudo** and **su**, as well as preventing them from executing files and applications in their home directory. If configured, this prevents users from executing malicious files from their home directories.
- Process separation is used. Processes run in their own domains, preventing processes from accessing files used by other processes, as well as preventing processes from accessing other processes. For example, when running SELinux, unless otherwise configured, an attacker cannot compromise a Samba server, and then use that Samba server as an attack vector to read and write to files used by other processes, such as databases used by MySQL.
- SELinux helps limit the damage made by configuration mistakes. Domain Name System (DNS) servers often replicate information between each other in what is known as a zone transfer. Attackers can use zone transfers to update DNS servers with false information. When running the Berkeley Internet Name Domain (BIND) as a DNS server in Red Hat Enterprise Linux, even if an administrator forgets to limit which servers can perform a zone transfer, the default SELinux policy prevents zone files [3] from being updated via zone transfers, by the BIND **named** daemon itself, and by other processes.
- Refer to the Red Hat Magazine article, Risk report: Three years of Red Hat Enterprise Linux 4 [4], for exploits that were restricted due to the default SELinux targeted policy in Red Hat Enterprise Linux 4.
- Refer to the NetworkWorld.com article, A seatbelt for server software: SELinux blocks real-world exploits [5], for background information about SELinux, and information about various exploits that SELinux has prevented.
- Refer to James Morris's SELinux mitigates remote root vulnerability in OpenPegasus blog post for information about an exploit in OpenPegasus that was mitigated by SELinux as shipped with Red Hat Enterprise Linux 4 and 5.

2.3. SELinux Architecture

SELinux is a Linux security module that is built into the Linux kernel. SELinux is driven by loadable policy rules. When security-relevant access is taking place, such as when a process attempts to open a file, the operation is intercepted in the kernel by SELinux. If an SELinux policy rule allows the operation, it continues, otherwise, the operation is blocked and the process receives an error.

SELinux decisions, such as allowing or disallowing access, are cached. This cache is known as the Access Vector Cache (AVC). When using these cached decisions, SELinux policy rules need to be checked less, which increases performance. Remember that SELinux policy rules have no effect if DAC rules deny access first.

2.4. SELinux Modes

SELinux has three modes:

- Enforcing: SELinux policy is enforced. SELinux denies access based on SELinux policy rules.
- Permissive: SELinux policy is not enforced. SELinux does not deny access, but denials are logged for actions that would have been denied if running in enforcing mode.
- Disabled: SELinux is disabled. Only DAC rules are used.

Use the **setenforce** command to change between enforcing and permissive mode. Changes made with **setenforce** do not persist across reboots. To change to enforcing mode, as the Linux root user, run the **setenforce 1** command. To change to permissive mode, run the **setenforce 0** command. Use the **getenforce** command to view the current SELinux mode.

Persistent mode changes are covered in Section 5.4, “Enabling and Disabling SELinux”.

---

[1] "Integrating Flexible Support for Security Policies into the Linux Operating System", by Peter Loscocco and Stephen Smalley. This paper was originally prepared for the National Security Agency and is, consequently, in the public domain. Refer to the original paper for details and the document as it was first released. Any edits and changes were done by Murray McAllister.

[2] "Meeting Critical Security Objectives with Security-Enhanced Linux", by Peter Loscocco and Stephen Smalley. This paper was originally prepared for the National Security Agency and is, consequently, in the public domain. Refer to the original paper for details and the document as it was first released. Any edits and changes were done by Murray McAllister.

[3] Text files that include information, such as host name to IP address mappings, that are used by DNS servers.

[4] Cox, Mark. "Risk report: Three years of Red Hat Enterprise Linux 4". Published 26 February 2008. Accessed 27 August 2009: http://magazine.redhat.com/2008/02/26/risk-report-three-years-of-red-hat-enterprise-linux-4/.

[5] Marti, Don. "A seatbelt for server software: SELinux blocks real-world exploits". Published 24 February 2008. Accessed 27 August 2009: http://www.networkworld.com/news/2008/022408-selinux.html.

Chapter 3. SELinux Contexts

Processes and files are labeled with an SELinux context that contains additional information, such as an SELinux user, role, type, and, optionally, a level. When running SELinux, all of this information is used to make access control decisions. In Red Hat Enterprise Linux, SELinux provides a combination of Role-Based Access Control (RBAC), Type Enforcement (TE), and, optionally, Multi-Level Security (MLS).

The following is an example showing SELinux context. SELinux contexts are used on processes, Linux users, and files, on Linux operating systems that run SELinux. Use the **`ls -Z`** command to view the SELinux context of files and directories:

```
~]$ ls -Z file1
-rwxrw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0      file1
```

SELinux contexts follow the *SELinux user:role:type:level* syntax. The fields are as follows:

SELinux user

The SELinux user identity is an identity known to the policy that is authorized for a specific set of roles, and for a specific MLS/MCS range. Each Linux user is mapped to an SELinux user via SELinux policy. This allows Linux users to inherit the restrictions placed on SELinux users. The mapped SELinux user identity is used in the SELinux context for processes in that session, in order to define what roles and levels they can enter. Run the **`semanage login -l`** command as the Linux root user to view a list of mappings between SELinux and Linux user accounts (you need to have the *policycoreutils-python* package installed):

```
~]# semanage login -l

Login Name                SELinux User              MLS/MCS Range

__default__               unconfined_u              s0-s0:c0.c1023
root                      unconfined_u              s0-
s0:c0.c1023
system_u                  system_u                  s0-s0:c0.c1023
```

Output may differ slightly from system to system. The **`Login Name`** column lists Linux users, and the **`SELinux User`** column lists which SELinux user the Linux user is mapped to. For processes, the SELinux user limits which roles and levels are accessible. The last column, **`MLS/MCS Range`**, is the level used by Multi-Level Security (MLS) and Multi-Category Security (MCS).

role

Part of SELinux is the Role-Based Access Control (RBAC) security model. The role is an attribute of RBAC. SELinux users are authorized for roles, and roles are authorized for domains. The role serves as an intermediary between domains and SELinux users. The roles that can be entered determine which domains can be entered; ultimately, this controls which object types can be accessed. This helps reduce vulnerability to privilege escalation attacks.

type

The type is an attribute of Type Enforcement. The type defines a domain for processes, and a type for files. SELinux policy rules define how types can access each other, whether it be a domain accessing a type, or a domain accessing another domain. Access is only allowed if a specific SELinux policy rule exists that allows it.

level

The level is an attribute of MLS and MCS. An MLS range is a pair of levels, written as *lowlevel-highlevel* if the levels differ, or *lowlevel* if the levels are identical (**s0-s0** is the same as **s0**). Each level is a sensitivity-category pair, with categories being optional. If there are categories, the level is written as *sensitivity:category-set*. If there are no categories, it is written as *sensitivity*.

If the category set is a contiguous series, it can be abbreviated. For example, **c0.c3** is the same as **c0,c1,c2,c3**. The **/etc/selinux/targeted/setrans.conf** file maps levels (**s0:c0**) to human-readable form (that is **CompanyConfidential**). Do not edit **setrans.conf** with a text editor: use the **semanage** command to make changes. Refer to the semanage(8) manual page for further information. In Red Hat Enterprise Linux, targeted policy enforces MCS, and in MCS, there is just one sensitivity, **s0**. MCS in Red Hat Enterprise Linux supports 1024 different categories: **c0** through to **c1023**. **s0-s0:c0.c1023** is sensitivity **s0** and authorized for all categories.

MLS enforces the Bell-La Padula Mandatory Access Model, and is used in Labeled Security Protection Profile (LSPP) environments. To use MLS restrictions, install the *selinux-policy-mls* package, and configure MLS to be the default SELinux policy. The MLS policy shipped with Red Hat Enterprise Linux omits many program domains that were not part of the evaluated configuration, and therefore, MLS on a desktop workstation is unusable (no support for the X Window System); however, an MLS policy from the upstream SELinux Reference Policy can be built that includes all program domains. For more information on MLS configuration, refer to Section 5.11, “Multi-Level Security (MLS)”.

3.1. Domain Transitions

A process in one domain transitions to another domain by executing an application that has the **entrypoint** type for the new domain. The **entrypoint** permission is used in SELinux policy, and controls which applications can be used to enter a domain. The following example demonstrates a domain transition:

1. A user wants to change their password. To do this, they run the **passwd** application. The **/usr/bin/passwd** executable is labeled with the **passwd_exec_t** type:

```
~]$ ls -Z /usr/bin/passwd
-rwsr-xr-x  root root system_u:object_r:passwd_exec_t:s0
/usr/bin/passwd
```

   The **passwd** application accesses **/etc/shadow**, which is labeled with the **shadow_t** type:

```
~]$ ls -Z /etc/shadow
-r--------. root root system_u:object_r:shadow_t:s0    /etc/shadow
```

2. An SELinux policy rule states that processes running in the **passwd_t** domain are allowed to read and write to files labeled with the **shadow_t** type. The **shadow_t** type is only applied to files that are required for a password change. This includes **/etc/gshadow**, **/etc/shadow**, and their backup files.

3. An SELinux policy rule states that the **passwd_t** domain has **entrypoint** permission to the **passwd_exec_t** type.

4. When a user runs the **passwd** application, the user's shell process transitions to the **passwd_t** domain. With SELinux, since the default action is to deny, and a rule exists that allows (among other things) applications running in the **passwd_t** domain to access files labeled with the **shadow_t** type, the **passwd** application is allowed to access **/etc/shadow**, and update the user's password.

This example is not exhaustive, and is used as a basic example to explain domain transition. Although there is an actual rule that allows subjects running in the **passwd_t** domain to access objects labeled with the **shadow_t** file type, other SELinux policy rules must be met before the subject can transition to a new domain. In this example, Type Enforcement ensures:

- The **passwd_t** domain can only be entered by executing an application labeled with the **passwd_exec_t** type; can only execute from authorized shared libraries, such as the **lib_t** type; and cannot execute any other applications.
- Only authorized domains, such as **passwd_t**, can write to files labeled with the **shadow_t** type. Even if other processes are running with superuser privileges, those processes cannot write to files labeled with the **shadow_t** type, as they are not running in the **passwd_t** domain.
- Only authorized domains can transition to the **passwd_t** domain. For example, the **sendmail** process running in the **sendmail_t** domain does not have a legitimate reason to execute **passwd**; therefore, it can never transition to the **passwd_t** domain.
- Processes running in the **passwd_t** domain can only read and write to authorized types, such as files labeled with the **etc_t** or **shadow_t** types. This prevents the **passwd** application from being tricked into reading or writing arbitrary files.

3.2. SELinux Contexts for Processes

Use the **ps -eZ** command to view the SELinux context for processes. For example:

1. Open a terminal, such as **Applications** → **System Tools** → **Terminal**.
2. Run the **passwd** command. Do not enter a new password.
3. Open a new tab, or another terminal, and run the **ps -eZ | grep passwd** command. The output is similar to the following:

```
unconfined_u:unconfined_r:passwd_t:s0-s0:c0.c1023 13212 pts/1
00:00:00 passwd
```

4. In the first tab/terminal, press **Ctrl+C** to cancel the **passwd** application.

In this example, when the **passwd** application (labeled with the **passwd_exec_t** type) is executed, the user's shell process transitions to the **passwd_t** domain. Remember that the type defines a domain for processes, and a type for files.

Use the **ps -eZ** command to view the SELinux contexts for running processes. The following is a truncated example of the output, and may differ on your system:

```
system_u:system_r:dhcpc_t:s0              1869 ?  00:00:00 dhclient
system_u:system_r:sshd_t:s0-s0:c0.c1023  1882 ?  00:00:00 sshd
system_u:system_r:gpm_t:s0                1964 ?  00:00:00 gpm
system_u:system_r:crond_t:s0-s0:c0.c1023 1973 ?  00:00:00 crond
```

```
system_u:system_r:kerneloops_t:s0        1983 ?  00:00:05 kerneloops
system_u:system_r:crond_t:s0-s0:c0.c1023 1991 ?  00:00:00 atd
```

The **system_r** role is used for system processes, such as daemons. Type Enforcement then separates each domain.

3.3. SELinux Contexts for Users

Use the **id -Z** command to view the SELinux context associated with your Linux user:

```
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

In Red Hat Enterprise Linux, Linux users run unconfined by default. This SELinux context shows that the Linux user is mapped to the SELinux **unconfined_u** user, running as the **unconfined_r** role, and is running in the **unconfined_t** domain. **s0-s0** is an MLS range, which in this case, is the same as just **s0**. The categories the user has access to is defined by **c0.c1023**, which is all categories (**c0** through to **c1023**).

Chapter 4. Targeted Policy

Targeted policy is the default SELinux policy used in Red Hat Enterprise Linux. When using targeted policy, processes that are targeted run in a confined domain, and processes that are not targeted run in an unconfined domain. For example, by default, logged-in users run in the **unconfined_t** domain, and system processes started by init run in the **initrc_t** domain; both of these domains are unconfined.

Unconfined domains (as well as confined domains) are subject to executable and writeable memory checks. By default, subjects running in an unconfined domain cannot allocate writeable memory and execute it. This reduces vulnerability to buffer overflow attacks. These memory checks are disabled by setting Booleans, which allow the SELinux policy to be modified at runtime. Boolean configuration is discussed later.

4.1. Confined Processes

Almost every service that listens on a network, such as **sshd** or **httpd**, is confined in Red Hat Enterprise Linux. Also, most processes that run as the Linux root user and perform tasks for users, such as the **passwd** application, are confined. When a process is confined, it runs in its own domain, such as the **httpd** process running in the **httpd_t** domain. If a confined process is compromised by an attacker, depending on SELinux policy configuration, an attacker's access to resources and the possible damage they can do is limited.

Complete this procedure to ensure that SELinux is enabled and the system is prepared to perform the following example:

Procedure 4.1. How to Verify SELinux Status

1. Run the **sestatus** command to confirm that SELinux is enabled, is running in enforcing mode, and that targeted policy is being used. The correct output should look similar to the output bellow.

```
~]$ sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   enforcing
Mode from config file:          enforcing
Policy version:                 24
Policy from config file:        targeted
```

   Refer to the section Section 5.4, “Enabling and Disabling SELinux” for detailed information about enabling and disabling SELinux.

2. As the Linux root user, run the **touch /var/www/html/testfile** command to create a file.

3. Run the **ls -Z /var/www/html/testfile** command to view the SELinux context:

```
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
/var/www/html/testfile
```

   By default, Linux users run unconfined in Red Hat Enterprise Linux, which is why the **testfile** file is labeled with the SELinux **unconfined_u** user. RBAC is used for processes, not files. Roles do not have a meaning for files; the **object_r** role is a generic role used for files (on persistent storage and network file systems). Under the **/proc/** directory, files related

to processes may use the **system_r** role. [6] The **httpd_sys_content_t** type allows the **httpd** process to access this file.

The following example demonstrates how SELinux prevents the Apache HTTP Server (**httpd**) from reading files that are not correctly labeled, such as files intended for use by Samba. This is an example, and should not be used in production. It assumes that the *httpd* and *wget* packages are installed, the SELinux targeted policy is used, and that SELinux is running in enforcing mode.

Procedure 4.2. An Example of Confined Process

1. As the Linux root user, run the **service httpd start** command to start the **httpd** process. The output is as follows if **httpd** starts successfully:

```
~]# service httpd start
Starting httpd:                                            [  OK  ]
```

2. Change into a directory where your Linux user has write access to, and run the **wget http://localhost/testfile** command. Unless there are changes to the default configuration, this command succeeds:

```
~]$ wget http://localhost/testfile
--2009-11-06 17:43:01--  http://localhost/testfile
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 0 [text/plain]
Saving to: `testfile'

[ <=>                                 ] 0     --.-K/s   in 0s

2009-11-06 17:43:01 (0.00 B/s) - `testfile' saved [0/0]
```

3. The **chcon** command relabels files; however, such label changes do not survive when the file system is relabeled. For permanent changes that survive a file system relabel, use the **semanage** command, which is discussed later. As the Linux root user, run the following command to change the type to a type used by Samba:

```
~]# chcon -t samba_share_t /var/www/html/testfile
```

Run the **ls -Z /var/www/html/testfile** command to view the changes:

```
-rw-r--r--  root root unconfined_u:object_r:samba_share_t:s0
/var/www/html/testfile
```

4. Note: the current DAC permissions allow the **httpd** process access to **testfile**. Change into a directory where your Linux user has write access to, and run the **wget http://localhost/testfile** command. Unless there are changes to the default configuration, this command fails:

```
~]$ wget http://localhost/testfile
--2009-11-06 14:11:23--  http://localhost/testfile
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:80... connected.
```

```
HTTP request sent, awaiting response... 403 Forbidden
2009-11-06 14:11:23 ERROR 403: Forbidden.
```

5. As the Linux root user, run the **rm -i /var/www/html/testfile** command to remove **testfile**.

6. If you do not require **httpd** to be running, as the Linux root user, run the **service httpd stop** command to stop **httpd**:

```
~]# service httpd stop
Stopping httpd:                                            [  OK  ]
```

This example demonstrates the additional security added by SELinux. Although DAC rules allowed the **httpd** process access to **testfile** in step 2, because the file was labeled with a type that the **httpd** process does not have access to, SELinux denied access.

If the **auditd** daemon is running, an error similar to the following is logged to **/var/log/audit/audit.log**:

```
type=AVC msg=audit(1220706212.937:70): avc:  denied  { getattr } for
pid=1904 comm="httpd" path="/var/www/html/testfile" dev=sda5 ino=247576
scontext=unconfined_u:system_r:httpd_t:s0
tcontext=unconfined_u:object_r:samba_share_t:s0  tclass=file

type=SYSCALL msg=audit(1220706212.937:70): arch=40000003 syscall=196
success=no exit=-13 a0=b9e21da0 a1=bf9581dc a2=555ff4 a3=2008171 items=0
ppid=1902 pid=1904 auid=500 uid=48 gid=48 euid=48 suid=48 fsuid=48
egid=48 sgid=48 fsgid=48 tty=(none) ses=1 comm="httpd"
exe="/usr/sbin/httpd" subj=unconfined_u:system_r:httpd_t:s0 key=(null)
```

Also, an error similar to the following is logged to **/var/log/httpd/error_log**:

```
[Wed May 06 23:00:54 2009] [error] [client 127.0.0.1] (13)Permission
denied: access to /testfile denied
```

4.2. Unconfined Processes

Unconfined processes run in unconfined domains, for example, init programs run in the unconfined **initrc_t** domain, unconfined kernel processes run in the **kernel_t** domain, and unconfined Linux users run in the **unconfined_t** domain. For unconfined processes, SELinux policy rules are applied, but policy rules exist that allow processes running in unconfined domains almost all access. Processes running in unconfined domains fall back to using DAC rules exclusively. If an unconfined process is compromised, SELinux does not prevent an attacker from gaining access to system resources and data, but of course, DAC rules are still used. SELinux is a security enhancement on top of DAC rules – it does not replace them.

To ensure that SELinux is enabled and the system is prepared to perform the following example, complete the Procedure 4.1, “How to Verify SELinux Status” described in Section 4.1, “Confined Processes”.

The following example demonstrates how the Apache HTTP Server (**httpd**) can access data intended for use by Samba, when running unconfined. Note that in Red Hat Enterprise Linux, the **httpd** process runs in the confined **httpd_t** domain by default. This is an example, and should not be used in production. It assumes that the *httpd*, *wget*, *dbus* and *audit* packages are installed, that the

SELinux targeted policy is used, and that SELinux is running in enforcing mode.

Procedure 4.3. An Example of Unconfined Process

1. The **chcon** command relabels files; however, such label changes do not survive when the file system is relabeled. For permanent changes that survive a file system relabel, use the **semanage** command, which is discussed later. As the Linux root user, run the following command to change the type to a type used by Samba:

```
~]# chcon -t samba_share_t /var/www/html/testfile
```

   Run the **ls -Z /var/www/html/testfile** command to view the changes:

```
~]$ ls -Z /var/www/html/testfile
-rw-r--r--  root root unconfined_u:object_r:samba_share_t:s0
/var/www/html/testfile
```

2. Run the **service httpd status** command to confirm that the **httpd** process is not running:

```
~]$ service httpd status
httpd is stopped
```

   If the output differs, run the **service httpd stop** command as the Linux root user to stop the **httpd** process:

```
~]# service httpd stop
Stopping httpd:                                            [  OK  ]
```

3. To make the **httpd** process run unconfined, run the following command as the Linux root user to change the type of **/usr/sbin/httpd**, to a type that does not transition to a confined domain:

```
~]# chcon -t unconfined_exec_t /usr/sbin/httpd
```

4. Run the **ls -Z /usr/sbin/httpd** command to confirm that **/usr/sbin/httpd** is labeled with the **unconfined_exec_t** type:

```
~]$ ls -Z /usr/sbin/httpd
-rwxr-xr-x  root root system_u:object_r:unconfined_exec_t:s0
/usr/sbin/httpd
```

5. As the Linux root user, run the **service httpd start** command to start the **httpd** process. The output is as follows if **httpd** starts successfully:

```
~]# service httpd start
Starting httpd:                                            [  OK  ]
```

6. Run the **ps -eZ | grep httpd** command to view the **httpd** running in the **unconfined_t** domain:

```
~]$ ps -eZ | grep httpd
unconfined_u:unconfined_r:unconfined_t:s0 7721 ?      00:00:00
```

```
httpd
unconfined_u:unconfined_r:unconfined_t:s0 7723 ?      00:00:00
httpd
unconfined_u:unconfined_r:unconfined_t:s0 7724 ?      00:00:00
httpd
unconfined_u:unconfined_r:unconfined_t:s0 7725 ?      00:00:00
httpd
unconfined_u:unconfined_r:unconfined_t:s0 7726 ?      00:00:00
httpd
unconfined_u:unconfined_r:unconfined_t:s0 7727 ?      00:00:00
httpd
unconfined_u:unconfined_r:unconfined_t:s0 7728 ?      00:00:00
httpd
unconfined_u:unconfined_r:unconfined_t:s0 7729 ?      00:00:00
httpd
unconfined_u:unconfined_r:unconfined_t:s0 7730 ?      00:00:00
httpd
```

7. Change into a directory where your Linux user has write access to, and run the **wget http://localhost/testfile** command. Unless there are changes to the default configuration, this command succeeds:

```
~]$ wget http://localhost/testfile
--2009-05-07 01:41:10--  http://localhost/testfile
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 0 [text/plain]
Saving to: `testfile.1'

[ <=>                                 ]--.-K/s   in 0s

2009-05-07 01:41:10 (0.00 B/s) - `testfile.1' saved [0/0]
```

Although the **httpd** process does not have access to files labeled with the **samba_share_t** type, **httpd** is running in the unconfined **unconfined_t** domain, and falls back to using DAC rules, and as such, the **wget** command succeeds. Had **httpd** been running in the confined **httpd_t** domain, the **wget** command would have failed.

8. The **restorecon** command restores the default SELinux context for files. As the Linux root user, run the **restorecon -v /usr/sbin/httpd** command to restore the default SELinux context for **/usr/sbin/httpd**:

```
~]# restorecon -v /usr/sbin/httpd
restorecon reset /usr/sbin/httpd context
system_u:object_r:unconfined_exec_t:s0-
>system_u:object_r:httpd_exec_t:s0
```

Run the **ls -Z /usr/sbin/httpd** command to confirm that **/usr/sbin/httpd** is labeled with the **httpd_exec_t** type:

```
~]$ ls -Z /usr/sbin/httpd
-rwxr-xr-x  root root system_u:object_r:httpd_exec_t:s0
/usr/sbin/httpd
```

9. As the Linux root user, run the **service httpd restart** command to restart **httpd**. After restarting, run the **ps -eZ | grep httpd** command to confirm that **httpd** is running in the confined **httpd_t** domain:

```
~]# service httpd restart
Stopping httpd:                                            [  OK  ]
Starting httpd:                                            [  OK  ]
~]# ps -eZ | grep httpd
unconfined_u:system_r:httpd_t:s0    8883 ?        00:00:00 httpd
unconfined_u:system_r:httpd_t:s0    8884 ?        00:00:00 httpd
unconfined_u:system_r:httpd_t:s0    8885 ?        00:00:00 httpd
unconfined_u:system_r:httpd_t:s0    8886 ?        00:00:00 httpd
unconfined_u:system_r:httpd_t:s0    8887 ?        00:00:00 httpd
unconfined_u:system_r:httpd_t:s0    8888 ?        00:00:00 httpd
unconfined_u:system_r:httpd_t:s0    8889 ?        00:00:00 httpd
```

10. As the Linux root user, run the **rm -i /var/www/html/testfile** command to remove **testfile**:

```
~]# rm -i /var/www/html/testfile
rm: remove regular empty file `/var/www/html/testfile'? y
```

11. If you do not require **httpd** to be running, as the Linux root user, run the **service httpd stop** command to stop **httpd**:

```
~]# service httpd stop
Stopping httpd:                                            [  OK  ]
```

The examples in these sections demonstrate how data can be protected from a compromised confined-process (protected by SELinux), as well as how data is more accessible to an attacker from a compromised unconfined-process (not protected by SELinux).

4.3. Confined and Unconfined Users

Each Linux user is mapped to an SELinux user via SELinux policy. This allows Linux users to inherit the restrictions on SELinux users. This Linux user mapping is seen by running the **semanage login -l** command as the Linux root user:

```
~]# semanage login -l

Login Name                SELinux User              MLS/MCS Range

__default__               unconfined_u              s0-s0:c0.c1023
root                      unconfined_u              s0-s0:c0.c1023
system_u                  system_u                  s0-s0:c0.c1023
```

In Red Hat Enterprise Linux 6, Linux users are mapped to the SELinux **__default__** login by default, which is mapped to the SELinux **unconfined_u** user. The following line defines the default mapping:

```
__default__               unconfined_u              s0-s0:c0.c1023
```

The following procedure demonstrates how to add a new Linux user to the system and how to map that user to the SELinux **unconfined_u** user. It assumes that the Linux root user is running unconfined, as it does by default in Red Hat Enterprise Linux 6:

1. As the Linux root user, run the **useradd newuser** command to create a new Linux user named **newuser**.

2. As the Linux root user, run the **passwd newuser** command to assign a password to the Linux **newuser** user:

```
~]# passwd newuser
Changing password for user newuser.
New UNIX password: Enter a password
Retype new UNIX password: Enter the same password again
passwd: all authentication tokens updated successfully.
```

3. Log out of your current session, and log in as the Linux **newuser** user. When you log in, the **pam_selinux** PAM module automatically maps the Linux user to an SELinux user (in this case, **unconfined_u**), and sets up the resulting SELinux context. The Linux user's shell is then launched with this context. Run the **id -Z** command to view the context of a Linux user:

```
[newuser@localhost ~]$ id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Note

If you no longer need the **newuser** user on your system, log out of the Linux **newuser**'s session, log in with your account, and run the **userdel -r newuser** command as the Linux root user. It will remove **newuser** along with their home directory.

Confined and unconfined Linux users are subject to executable and writeable memory checks, and are also restricted by MCS or MLS.

If an unconfined Linux user executes an application that SELinux policy defines as one that can transition from the **unconfined_t** domain to its own confined domain, the unconfined Linux user is still subject to the restrictions of that confined domain. The security benefit of this is that, even though a Linux user is running unconfined, the application remains confined. Therefore, the exploitation of a flaw in the application can be limited by the policy.

Similarly, we can apply these checks to confined users. However, each confined Linux user is restricted by a confined user domain against the **unconfined_t** domain. The SELinux policy can also define a transition from a confined user domain to its own target confined domain. In such a case, confined Linux users are subject to the restrictions of that target confined domain. The main point is that special privileges are associated with the confined users according to their role. In the table below, you can see examples of basic confined domains for Linux users in Red Hat Enterprise Linux 6:

Table 4.1. SELinux User Capabilities

User	Domain	X Window System	su or sudo	Execute in home directory and /tmp/ (default)	Networking
sysadm_u	sysadm_t	yes	**su** and **sudo**	yes	yes
staff_u	staff_t	yes	only **sudo**	yes	yes
user_u	user_t	yes	no	yes	yes
guest_u	guest_t	no	no	no	yes
xguest_u	xguest_t	yes	no	no	Firefox only

- Linux users in the **user_t**, **guest_t**, **xguest_t**, and **git_shell_t** domains can only run set user ID (setuid) applications if SELinux policy permits it (for example, **passwd**). These users cannot run the **su** and **sudo** setuid applications, and therefore cannot use these applications to become the Linux root user.
- Linux users in the **sysadm_t**, **staff_t**, **user_t**, and **xguest_t** domains can log in via the X Window System and a terminal.
- By default, Linux users in the **guest_t** and **xguest_t** domains cannot execute applications in their home directories or **/tmp/**, preventing them from executing applications, which inherit users' permissions, in directories they have write access to. This helps prevent flawed or malicious applications from modifying users' files.
- By default, Linux users in the **staff_t** and **user_t** domains can execute applications in their home directories and **/tmp/**. Refer to Section 6.6, “Booleans for Users Executing Applications” for information about allowing and preventing users from executing applications in their home directories and **/tmp/**.
- The only network access Linux users in the **xguest_t** domain have is **Firefox** connecting to web pages.

[6] When using other policies, such as MLS, other roles may be used, for example, **secadm_r**.

Chapter 5. Working with SELinux

The following sections give a brief overview of the main SELinux packages in Red Hat Enterprise Linux; installing and updating packages; which log files are used; the main SELinux configuration file; enabling and disabling SELinux; SELinux modes; configuring Booleans; temporarily and persistently changing file and directory labels; overriding file system labels with the **mount** command; mounting NFS volumes; and how to preserve SELinux contexts when copying and archiving files and directories.

5.1. SELinux Packages

In Red Hat Enterprise Linux, the SELinux packages are installed by default, in a full installation, unless they are manually excluded during installation. If performing a minimal installation in text mode, the *policycoreutils-python* and the *policycoreutils-gui* package are not installed by default. Also, by default, SELinux targeted policy is used, and SELinux runs in enforcing mode. The following is a brief description of the SELinux packages that are installed on your system by default:

- *policycoreutils* provides utilities such as **restorecon**, **secon**, **setfiles**, **semodule**, **load_policy**, and **setsebool**, for operating and managing SELinux.
- *selinux-policy* provides the SELinux Reference Policy. The SELinux Reference Policy is a complete SELinux policy, and is used as a basis for other policies, such as the SELinux targeted policy; refer to the Tresys Technology SELinux Reference Policy page for further information. This package also provides the **/usr/share/selinux/devel/policygentool** development utility, as well as example policy files.
- *selinux-policy-targeted* provides the SELinux targeted policy.
- *libselinux* – provides an API for SELinux applications.
- *libselinux-utils* provides the **avcstat**, **getenforce**, **getsebool**, **matchpathcon**, **selinuxconlist**, **selinuxdefcon**, **selinuxenabled**, **setenforce**, and **togglesebool** utilities.
- *libselinux-python* provides Python bindings for developing SELinux applications.

The following is a brief description of the main optional packages, which have to be installed via the **yum install <*package-name*>** command:

- *selinux-policy-mls* provides the MLS SELinux policy.
- *setroubleshoot-server* translates denial messages, produced when access is denied by SELinux, into detailed descriptions that are viewed with the **sealert** utility, also provided by this package.
- *setools-console* – this package provides the Tresys Technology SETools distribution, a number of tools and libraries for analyzing and querying policy, audit log monitoring and reporting, and file context management [7]. The *setools* package is a meta-package for SETools. The *setools-gui* package provides the **apol**, **seaudit**, and **sediffx** tools. The *setools-console* package provides the **seaudit-report**, **sechecker**, **sediff**, **seinfo**, **sesearch**, **findcon**, **replcon**, and **indexcon** command-line tools. Refer to the Tresys Technology SETools page for information about these tools.
- *mcstrans* translates levels, such as **s0-s0:c0.c1023**, to an easier to read form, such as **SystemLow-SystemHigh**. This package is not installed by default.
- *policycoreutils-python* provides utilities such as **semanage**, **audit2allow**, **audit2why**, and **chcat**, for operating and managing SELinux.

- *policycoreutils-gui* provides **system-config-selinux**, a graphical tool for managing SELinux.

5.2. Which Log File is Used

In Red Hat Enterprise Linux 6, the *dbus* and *audit* packages are installed by default, unless they are removed from the default package selection. The *setroubleshoot-server* must be installed via Yum (the **yum install setroubleshoot** command).

If the **auditd** daemon is running, SELinux denial messages, such as the following, are written to **/var/log/audit/audit.log** by default:

```
type=AVC msg=audit(1223024155.684:49): avc:  denied  { getattr } for
pid=2000 comm="httpd" path="/var/www/html/file1" dev=dm-0 ino=399185
scontext=unconfined_u:system_r:httpd_t:s0
tcontext=system_u:object_r:samba_share_t:s0 tclass=file
```

```
May 7 18:55:56 localhost setroubleshoot: SELinux is preventing httpd
(httpd_t) "getattr" to /var/www/html/file1 (samba_share_t). For complete
SELinux messages. run sealert -l de7e30d6-5488-466d-a606-92c9f40d316d
```

In Red Hat Enterprise Linux 6, **setroubleshootd** no longer constantly runs as a service. However, it is still used to analyze the AVC messages. Two new programs act as a method to start **setroubleshoot** when needed: **sedispatch** and **seapplet**. The **sedispatch** utility runs as part of the audit subsystem, and via **dbus**, sends a message when an AVC denial message is returned. These messages go straight to **setroubleshootd** if it is already running. If **setroubleshootd** is not running, **sedispatch** starts it automatically. The **seapplet** utility runs in the system toolbar, waiting for dbus messages in **setroubleshootd**. It launches the notification bubble, allowing the user to review AVC messages.

Procedure 5.1. Starting Daemons Automatically

To configure the **auditd** and **rsyslogd** daemons to automatically start at boot, run the following commands as the Linux root user:

1. ```
   ~]# chkconfig --levels 2345 auditd on
   ~]# chkconfig --levels 2345 rsyslog on
   ```

2. Use the **service *service-name* status** command to check if these services are running, for example:

   ```
   ~]# service auditd status
   auditd (pid  1318) is running...
   ```

3. If the above services are not running (**service-name is stopped**), use the **service *service-name* start** command as the Linux root user to start them. For example:

   ```
   ~]# service auditd start
   Starting auditd:                                   [  OK  ]
   ```

5.3. Main Configuration File
   ```

The **/etc/selinux/config** file is the main SELinux configuration file. It controls the SELinux mode and the SELinux policy to use:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=enforcing
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

SELINUX=enforcing

The **SELINUX** option sets the mode SELinux runs in. SELinux has three modes: enforcing, permissive, and disabled. When using enforcing mode, SELinux policy is enforced, and SELinux denies access based on SELinux policy rules. Denial messages are logged. When using permissive mode, SELinux policy is not enforced. SELinux does not deny access, but denials are logged for actions that would have been denied if running SELinux in enforcing mode. When using disabled mode, SELinux is disabled (the SELinux module is not registered with the Linux kernel), and only DAC rules are used.

SELINUXTYPE=targeted

The **SELINUXTYPE** option sets the SELinux policy to use. Targeted policy is the default policy. Only change this option if you want to use the MLS policy. For information on how to enable the MLS policy, refer to Section 5.11.2, “Enabling MLS in SELinux”.

Important

When systems run with SELinux in permissive or disabled mode, users have permission to label files incorrectly. Also, files created while SELinux is disabled are not labeled. This causes problems when changing to enforcing mode. To prevent incorrectly labeled and unlabeled files from causing problems, file systems are automatically relabeled when changing from disabled mode to permissive or enforcing mode.

5.4. Enabling and Disabling SELinux

Use the **getenforce** or **sestatus** commands to check the status of SELinux. The **getenforce** command returns **Enforcing**, **Permissive**, or **Disabled**.

The **sestatus** command returns the SELinux status and the SELinux policy being used:

```
~]$ sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   enforcing
Mode from config file:          enforcing
Policy version:                 24
Policy from config file:        targeted
```

5.4.1. Enabling SELinux

Important

If the system was initially installed without SELinux, particularly the *selinux-policy* package, which was added to the system later, one additional step is necessary to enable SELinux. To make sure SELinux is initialized during system startup, the **dracut** utility has to be run to put SELinux awareness into the initramfs file system. Failing to do so causes SELinux not to start during system startup.

On systems with SELinux disabled, the **SELINUX=disabled** option is configured in **/etc/selinux/config**:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

Also, the **getenforce** command returns **Disabled**:

```
~]$ getenforce
Disabled
```

To enable SELinux:

1. Use the **rpm -qa | grep selinux**, **rpm -q policycoreutils**, and **rpm -qa | grep setroubleshoot** commands to confirm that the SELinux packages are installed. This guide assumes the following packages are installed: *selinux-policy-targeted*, *selinux-policy*, *libselinux*, *libselinux-python*, *libselinux-utils*, *policycoreutils*, *policycoreutils-python*, *setroubleshoot*, *setroubleshoot-server*, *setroubleshoot-plugins*. If these packages are not installed, as the Linux root user, install them via the **yum install *package-name*** command. The following packages are optional: *policycoreutils-gui*, *setroubleshoot*, and *mcstrans*.

2. Before SELinux is enabled, each file on the file system must be labeled with an SELinux context. Before this happens, confined domains may be denied access, preventing your system from booting correctly. To prevent this, configure **SELINUX=permissive** in **/etc/selinux/config**:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=permissive
```

```
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

3. As the Linux root user, run the **reboot** command to restart the system. During the next boot, file systems are labeled. The label process labels all files with an SELinux context:

```
*** Warning -- SELinux targeted policy relabel is required.
*** Relabeling could take a very long time, depending on file
*** system size and speed of hard drives.
****
```

   Each * (asterisk) character on the bottom line represents 1000 files that have been labeled. In the above example, four * characters represent 4000 files have been labeled. The time it takes to label all files depends upon the number of files on the system, and the speed of the hard disk drives. On modern systems, this process can take as little as 10 minutes.

4. In permissive mode, SELinux policy is not enforced, but denials are still logged for actions that would have been denied if running in enforcing mode. Before changing to enforcing mode, as the Linux root user, run the **grep "SELinux is preventing" /var/log/messages** command to confirm that SELinux did not deny actions during the last boot. If SELinux did not deny actions during the last boot, this command does not return any output. Refer to Chapter 8, *Troubleshooting* for troubleshooting information if SELinux denied access during boot.

5. If there were no denial messages in **/var/log/messages**, configure **SELINUX=enforcing** in **/etc/selinux/config**:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=enforcing
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

6. Reboot your system. After reboot, confirm that **getenforce** returns **Enforcing**:

```
~]$ getenforce
Enforcing
```

7. As the Linux root user, run the **semanage login -l** command to view the mapping between SELinux and Linux users. The output should be as follows:

```
Login Name                SELinux User              MLS/MCS Range

__default__               unconfined_u              s0-s0:c0.c1023
root                      unconfined_u              s0-s0:c0.c1023
system_u                  system_u                  s0-s0:c0.c1023
```

If this is not the case, run the following commands as the Linux root user to fix the user mappings. It is safe to ignore the **SELinux-user *username* is already defined** warnings if they occur, where *username* can be **unconfined_u**, **guest_u**, or **xguest_u**:

1. **semanage user -a -S targeted -P user -R "unconfined_r system_r" -r s0-s0:c0.c1023 unconfined_u**
2. **semanage login -m -S targeted -s "unconfined_u" -r s0-s0:c0.c1023 __default__**
3. **semanage login -m -S targeted -s "unconfined_u" -r s0-s0:c0.c1023 root**
4. **semanage user -a -S targeted -P user -R guest_r guest_u**
5. **semanage user -a -S targeted -P user -R xguest_r xguest_u**

Important

When systems run with SELinux in permissive or disabled mode, users have permission to label files incorrectly. Also, files created while SELinux is disabled are not labeled. This causes problems when changing to enforcing mode. To prevent incorrectly labeled and unlabeled files from causing problems, file systems are automatically relabeled when changing from disabled mode to permissive or enforcing mode.

5.4.2. Disabling SELinux

To disable SELinux, configure **SELINUX=disabled** in **/etc/selinux/config**:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

Reboot your system. After reboot, confirm that the **getenforce** command returns **Disabled**:

```
~]$ getenforce
Disabled
```

5.5. Booleans

Booleans allow parts of SELinux policy to be changed at runtime, without any knowledge of SELinux policy writing. This allows changes, such as allowing services access to NFS volumes, without reloading or recompiling SELinux policy.

5.5.1. Listing Booleans

For a list of Booleans, an explanation of what each one is, and whether they are on or off, run the **semanage boolean -l** command as the Linux root user. The following example does not list all Booleans:

```
~]# semanage boolean -l
SELinux boolean                          Description

ftp_home_dir                   -> off   Allow ftp to read and write files
in the user home directories
xen_use_nfs                    -> off   Allow xen to manage nfs files
xguest_connect_network         -> on    Allow xguest to configure Network
Manager
```

The **SELinux boolean** column lists Boolean names. The **Description** column lists whether the Booleans are on or off, and what they do.

In the following example, the **ftp_home_dir** Boolean is off, preventing the FTP daemon (**vsftpd**) from reading and writing to files in user home directories:

```
ftp_home_dir                   -> off   Allow ftp to read and write files
in the user home directories
```

The **getsebool -a** command lists Booleans, whether they are on or off, but does not give a description of each one. The following example does not list all Booleans:

```
~]$ getsebool -a
allow_console_login --> off
allow_cvs_read_shadow --> off
allow_daemons_dump_core --> on
```

Run the **getsebool *boolean-name*** command to only list the status of the *boolean-name* Boolean:

```
~]$ getsebool allow_console_login
allow_console_login --> off
```

Use a space-separated list to list multiple Booleans:

```
~]$ getsebool allow_console_login allow_cvs_read_shadow
allow_daemons_dump_core
allow_console_login --> off
allow_cvs_read_shadow --> off
allow_daemons_dump_core --> on
```

5.5.2. Configuring Booleans

Run the **setsebool** utility in the **setsebool *boolean_name* on/off** form to enable or disable Booleans.

The following example demonstrates configuring the **httpd_can_network_connect_db** Boolean:

1. By default, the **httpd_can_network_connect_db** Boolean is off, preventing Apache HTTP Server scripts and modules from connecting to database servers:

```
~]$ getsebool httpd_can_network_connect_db
httpd_can_network_connect_db --> off
```

2. To temporarily enable Apache HTTP Server scripts and modules to connect to database servers, run the **setsebool httpd_can_network_connect_db on** command as the Linux root user.

3. Use the **getsebool httpd_can_network_connect_db** command to verify the Boolean is enabled:

```
~]$ getsebool httpd_can_network_connect_db
httpd_can_network_connect_db --> on
```

   This allows Apache HTTP Server scripts and modules to connect to database servers.

4. This change is not persistent across reboots. To make changes persistent across reboots, run the **setsebool -P *boolean-name* on** command as the Linux root user: [8]

```
~]# setsebool -P httpd_can_network_connect_db on
```

5.6. SELinux Contexts – Labeling Files

On systems running SELinux, all processes and files are labeled in a way that represents security-relevant information. This information is called the SELinux context. For files, this is viewed using the **ls -Z** command:

```
~]$ ls -Z file1
-rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
```

In this example, SELinux provides a user (**unconfined_u**), a role (**object_r**), a type (**user_home_t**), and a level (**s0**). This information is used to make access control decisions. On DAC systems, access is controlled based on Linux user and group IDs. SELinux policy rules are checked after DAC rules. SELinux policy rules are not used if DAC rules deny access first.

There are multiple commands for managing the SELinux context for files, such as **chcon**, **semanage fcontext**, and **restorecon**.

5.6.1. Temporary Changes: chcon

The **chcon** command changes the SELinux context for files. However, changes made with the **chcon** command do not survive a file system relabel, or the execution of the **restorecon** command. SELinux policy controls whether users are able to modify the SELinux context for any given file. When using **chcon**, users provide all or part of the SELinux context to change. An incorrect file type is a common cause of SELinux denying access.

Quick Reference

- Run the **chcon -t *type file-name*** command to change the file type, where *type* is a type, such as **httpd_sys_content_t**, and *file-name* is a file or directory name.
- Run the **chcon -R -t *type directory-name*** command to change the type of the directory and its contents, where *type* is a type, such as **httpd_sys_content_t**, and *directory-name* is a directory name.

Procedure 5.2. Changing a File's or Directory's Type

The following procedure demonstrates changing the type, and no other attributes of the SELinux context. The example in this section works the same for directories, for example, if **file1** was a directory.

1. Run the **cd** command without arguments to change into your home directory.

2. Run the **touch file1** command to create a new file. Use the **ls -Z file1** command to view the SELinux context for **file1**:

```
~]$ ls -Z file1
-rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
```

   In this example, the SELinux context for **file1** includes the SELinux **unconfined_u** user, **object_r** role, **user_home_t** type, and the **s0** level. For a description of each part of the SELinux context, refer to Chapter 3, *SELinux Contexts*.

3. Run the **chcon -t samba_share_t file1** command to change the type to **samba_share_t**. The **-t** option only changes the type. View the change with **ls -Z file1**:

```
~]$ ls -Z file1
-rw-rw-r--  user1 group1 unconfined_u:object_r:samba_share_t:s0
file1
```

4. Use the **restorecon -v file1** command to restore the SELinux context for the **file1** file. Use the **-v** option to view what changes:

```
~]$ restorecon -v file1
restorecon reset file1 context
unconfined_u:object_r:samba_share_t:s0-
>system_u:object_r:user_home_t:s0
```

   In this example, the previous type, **samba_share_t**, is restored to the correct, **user_home_t** type. When using targeted policy (the default SELinux policy in Red Hat Enterprise Linux 6), the **restorecon** command reads the files in the **/etc/selinux/targeted/contexts/files/** directory, to see which SELinux context files should have.

Procedure 5.3. Changing a Directory and its Contents Types

The following example demonstrates creating a new directory, and changing the directory's file type (along with its contents) to a type used by the Apache HTTP Server. The configuration in this example is used if you want Apache HTTP Server to use a different document root (instead of **/var/www/html/**):

1. As the Linux root user, run the **mkdir /web** command to create a new directory, and then the **touch /web/file{1,2,3}** command to create 3 empty files (**file1**, **file2**, and **file3**). The **/web/** directory and files in it are labeled with the **default_t** type:

```
~]# ls -dZ /web
drwxr-xr-x  root root unconfined_u:object_r:default_t:s0 /web
~]# ls -lZ /web
-rw-r--r--  root root unconfined_u:object_r:default_t:s0 file1
-rw-r--r--  root root unconfined_u:object_r:default_t:s0 file2
-rw-r--r--  root root unconfined_u:object_r:default_t:s0 file3
```

2. As the Linux root user, run the **chcon -R -t httpd_sys_content_t /web/** command to change the type of the **/web/** directory (and its contents) to **httpd_sys_content_t**:

```
~]# chcon -R -t httpd_sys_content_t /web/
~]# ls -dZ /web/
drwxr-xr-x  root root unconfined_u:object_r:httpd_sys_content_t:s0
/web/
~]# ls -lZ /web/
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
file1
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
file2
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
file3
```

3. As the Linux root user, run the **restorecon -R -v /web/** command to restore the default SELinux contexts:

```
~]# restorecon -R -v /web/
restorecon reset /web context
unconfined_u:object_r:httpd_sys_content_t:s0-
>system_u:object_r:default_t:s0
restorecon reset /web/file2 context
unconfined_u:object_r:httpd_sys_content_t:s0-
>system_u:object_r:default_t:s0
restorecon reset /web/file3 context
unconfined_u:object_r:httpd_sys_content_t:s0-
>system_u:object_r:default_t:s0
restorecon reset /web/file1 context
unconfined_u:object_r:httpd_sys_content_t:s0-
>system_u:object_r:default_t:s0
```

Refer to the chcon(1) manual page for further information about **chcon**.

Note

Type Enforcement is the main permission control used in SELinux targeted policy. For the most part, SELinux users and roles can be ignored.

5.6.2. Persistent Changes: semanage fcontext

The **semanage fcontext** command is used to change the SELinux context of files. When using targeted policy, changes are written to files located in the **/etc/selinux/targeted/contexts/files/** directory:

- The **file_contexts** file specifies default contexts for many files, as well as contexts updated via **semanage fcontext**.
- The **file_contexts.local** file stores contexts to newly created files and directories not found in **file_contexts**.

Two utilities read these files. The **setfiles** utility is used when a file system is relabeled and the **restorecon** utility restores the default SELinux contexts. This means that changes made by **semanage fcontext** are persistent, even if the file system is relabeled. SELinux policy controls whether users are able to modify the SELinux context for any given file.

Quick Reference

To make SELinux context changes that survive a file system relabel:

1. Run the **semanage fcontext -a *options file-name|directory-name*** command, remembering to use the full path to the file or directory.
2. Run the **restorecon -v *file-name|directory-name*** command to apply the context changes.

Procedure 5.4. Changing a File's or Directory's Type

The following example demonstrates changing a file's type, and no other attributes of the SELinux context. This example works the same for directories, for instance if **file1** was a directory.

1. As the Linux root user, run the **touch /etc/file1** command to create a new file. By default, newly-created files in the **/etc/** directory are labeled with the **etc_t** type:

```
~]# ls -Z /etc/file1
-rw-r--r--  root root unconfined_u:object_r:etc_t:s0
/etc/file1
```

   Use the **ls -dZ *directory_name*** command to list information about a directory.

2. As the Linux root user, run the **semanage fcontext -a -t samba_share_t /etc/file1** command to change the **file1** type to **samba_share_t**. The **-a** option adds a new record, and the **-t** option defines a type (**samba_share_t**). Note that running this command does not directly change the type; **file1** is still labeled with the **etc_t** type:

```
~]# semanage fcontext -a -t samba_share_t /etc/file1
~]# ls -Z /etc/file1
-rw-r--r--  root root unconfined_u:object_r:etc_t:s0
/etc/file1
```

   The **semanage fcontext -a -t samba_share_t /etc/file1** command adds the following entry to **/etc/selinux/targeted/contexts/files/file_contexts.local**:

```
/etc/file1    unconfined_u:object_r:samba_share_t:s0
```

3. As the Linux root user, run the **restorecon -v /etc/file1** command to change the type. Because the **semanage** command added an entry to **file.contexts.local** for **/etc/file1**, the **restorecon** command changes the type to **samba_share_t**:

```
~]# restorecon -v /etc/file1
restorecon reset /etc/file1 context unconfined_u:object_r:etc_t:s0-
>system_u:object_r:samba_share_t:s0
```

Procedure 5.5. Changing a Directory and its Contents Types

The following example demonstrates creating a new directory, and changing the directory's file type (along with its contents) to a type used by Apache HTTP Server. The configuration in this example is used if you want Apache HTTP Server to use a different document root (instead of **/var/www/html/**):

1. As the Linux root user, run the **mkdir /web** command to create a new directory, and then the **touch /web/file{1,2,3}** command to create 3 empty files (**file1**, **file2**, and **file3**). The **/web/** directory and files in it are labeled with the **default_t** type:

```
~]# ls -dZ /web
drwxr-xr-x  root root unconfined_u:object_r:default_t:s0 /web
~]# ls -lZ /web
-rw-r--r--  root root unconfined_u:object_r:default_t:s0 file1
-rw-r--r--  root root unconfined_u:object_r:default_t:s0 file2
-rw-r--r--  root root unconfined_u:object_r:default_t:s0 file3
```

2. As the Linux root user, run the **semanage fcontext -a -t httpd_sys_content_t "/web(/.*)?"** command to change the type of the **/web/** directory and the files in it, to **httpd_sys_content_t**. The **-a** option adds a new record, and the **-t** option defines a type (httpd_sys_content_t). The **"/web(/.*)?"** regular expression causes the **semanage** command to apply changes to the **/web/** directory, as well as the files in it. Note that running this command does not directly change the type; **/web/** and files in it are still labeled with the **default_t** type:

```
~]# ls -dZ /web
drwxr-xr-x  root root unconfined_u:object_r:default_t:s0 /web
~]# ls -lZ /web
-rw-r--r--  root root unconfined_u:object_r:default_t:s0 file1
-rw-r--r--  root root unconfined_u:object_r:default_t:s0 file2
-rw-r--r--  root root unconfined_u:object_r:default_t:s0 file3
```

   The **semanage fcontext -a -t httpd_sys_content_t "/web(/.*)?"** command adds the following entry to **/etc/selinux/targeted/contexts/files/file_contexts.local**:

```
/web(/.*)?    system_u:object_r:httpd_sys_content_t:s0
```

3. As the Linux root user, run the **restorecon -R -v /web** command to change the type of the **/web/** directory, as well as all files in it. The **-R** is for recursive, which means all files and directories under the **/web/** directory are labeled with the **httpd_sys_content_t** type. Since the **semanage** command added an entry to **file.contexts.local** for **/web(/.*)?**, the **restorecon** command changes the types to **httpd_sys_content_t**:

```
~]# restorecon -R -v /web
restorecon reset /web context unconfined_u:object_r:default_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
restorecon reset /web/file2 context
unconfined_u:object_r:default_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
restorecon reset /web/file3 context
unconfined_u:object_r:default_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
restorecon reset /web/file1 context
unconfined_u:object_r:default_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
```

Note

By default, newly-created files and directories inherit the SELinux type of their parent directories. For example, when creating a new file in the **/etc/** directory that is labeled with the **etc_t** type, the new file inherits the same type:

```
~]$ ls -dZ - /etc/
drwxr-xr-x. root root system_u:object_r:etc_t:s0       /etc
```

```
~]# touch /etc/file1
```

```
~]# ls -lZ /etc/file1
-rw-r--r--. root root unconfined_u:object_r:etc_t:s0
/etc/file1
```

Procedure 5.6. Deleting an added Context

The following example demonstrates adding and removing an SELinux context. If the context is part of a regular expression, for example, **/web(/.*)?**, use quotation marks around the regular expression:

```
~]# semanage fcontext -d "/web(/.*)?"
```

1. To remove the context, as the Linux root user, run the **semanage fcontext -d *file-name|directory-name*** command, where *file-name|directory-name* is the first part in **file_contexts.local**. The following is an example of a context in **file_contexts.local**:

   ```
   /test    system_u:object_r:httpd_sys_content_t:s0
   ```

   With the first part being **/test**. To prevent the **/test/** directory from being labeled with the **httpd_sys_content_t** after running **restorecon**, or after a file system relabel, run the following command as the Linux root user to delete the context from **file_contexts.local**:

   ```
   ~]# semanage fcontext -d /test
   ```

2. As the Linux root user, use the **restorecon** utility to restore the default SELinux context.

Refer to the semanage(8) manual page for further information about **semanage**.

Important

When changing the SELinux context with **semanage fcontext -a**, use the full path to the file or directory to avoid files being mislabeled after a file system relabel, or after the **restorecon** command is run.

5.7. The file_t and default_t Types

5.7.1. The file_t and default_t types

When using a file system that supports extended attributes (EA), the **file_t** type is the default type for files that have not been assigned an EA value. This type is only used for this purpose and does not exist on correctly labeled file systems, because all files on a system running SELinux should have a proper SELinux context, and the **file_t** type is never used in file-context configuration [9].

The **default_t** type is used on files that do not match any pattern in file-context configuration, so that such files can be distinguished from files that do not have a context on disk, and generally are kept inaccessible to confined domains. For example, if you create a new top-level directory, such as **/mydirectory/**, this directory may be labeled with the **default_t** type. If services need access to this directory, you need to update the file-context configuration for this location. See Section 5.6.2, "Persistent Changes: semanage fcontext" for details on adding a context to the file-context configuration.

5.8. Mounting File Systems

By default, when a file system that supports extended attributes is mounted, the security context for each file is obtained from the *security.selinux* extended attribute of the file. Files in file systems that do not support extended attributes are assigned a single, default security context from the policy configuration, based on file system type.

Use the **mount -o context** command to override existing extended attributes, or to specify a different, default context for file systems that do not support extended attributes. This is useful if you do not trust a file system to supply the correct attributes, for example, removable media used in multiple systems. The **mount -o context** command can also be used to support labeling for file systems that do not support extended attributes, such as File Allocation Table (FAT) or NFS volumes. The context specified with the **context** is not written to disk: the original contexts are preserved, and are seen when mounting without a **context** option (if the file system had extended attributes in the first place).

For further information about file system labeling, refer to James Morris's "Filesystem Labeling in SELinux" article: http://www.linuxjournal.com/article/7426.

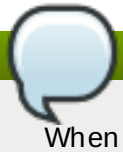
5.8.1. Context Mounts

To mount a file system with the specified context, overriding existing contexts if they exist, or to specify a different, default context for a file system that does not support extended attributes, as the Linux root user, use the **mount -o context=*SELinux_user:role:type:level*** command when mounting the desired file system. Context changes are not written to disk. By default, NFS mounts on the client side are labeled with a default context defined by policy for NFS volumes. In common policies, this default context uses the **nfs_t** type. Without additional mount options, this may prevent sharing NFS volumes via other services, such as the Apache HTTP Server. The following example mounts an NFS volume so that it can be shared via the Apache HTTP Server:

```
~]# mount server:/export /local/mount/point -o \
context="system_u:object_r:httpd_sys_content_t:s0"
```

Newly-created files and directories on this file system appear to have the SELinux context specified with **-o context**. However, since these changes are not written to disk, the context specified with this option does not persist between mounts. Therefore, this option must be used with the same context specified during every mount to retain the desired context. For information about making context mount persistent, refer to the Section 5.8.5, "Making Context Mounts Persistent".

Type Enforcement is the main permission control used in SELinux targeted policy. For the most part, SELinux users and roles can be ignored, so, when overriding the SELinux context with **-o context**, use the SELinux **system_u** user and **object_r** role, and concentrate on the type. If you are not using the MLS policy or multi-category security, use the **s0** level.



Note

When a file system is mounted with a **context** option, context changes (by users and processes) are prohibited. For example, running the **chcon** command on a file system mounted with a **context** option results in a **Operation not supported** error.

5.8.2. Changing the Default Context

As mentioned in Section 5.7, "The **file_t** and **default_t** Types", on file systems that support extended attributes, when a file that lacks an SELinux context on disk is accessed, it is treated as if it had a default context as defined by SELinux policy. In common policies, this default context uses the **file_t** type. If it is desirable to use a different default context, mount the file system with the **defcontext** option.

The following example mounts a newly-created file system (on **/dev/sda2**) to the newly-created **/test/** directory. It assumes that there are no rules in **/etc/selinux/targeted/contexts/files/** that define a context for the **/test/** directory:

```
~]# mount /dev/sda2 /test/ -o
defcontext="system_u:object_r:samba_share_t:s0"
```

In this example:

- the **defcontext** option defines that **system_u:object_r:samba_share_t:s0** is "the default security context for unlabeled files" [10].
- when mounted, the root directory (**/test/**) of the file system is treated as if it is labeled with the context specified by **defcontext** (this label is not stored on disk). This affects the labeling for files created under **/test/**: new files inherit the **samba_share_t** type, and these labels are stored on disk.
- files created under **/test/** while the file system was mounted with a **defcontext** option retain their labels.

5.8.3. Mounting an NFS Volume

By default, NFS mounts on the client side are labeled with a default context defined by policy for NFS volumes. In common policies, this default context uses the **nfs_t** type. Depending on policy configuration, services, such as Apache HTTP Server and MySQL, may not be able to read files labeled with the **nfs_t** type. This may prevent file systems labeled with this type from being mounted and then read or exported by other services.

If you would like to mount an NFS volume and read or export that file system with another service, use the **context** option when mounting to override the **nfs_t** type. Use the following context option to mount NFS volumes so that they can be shared via the Apache HTTP Server:

```
~]# mount server:/export /local/mount/point -o
context="system_u:object_r:httpd_sys_content_t:s0"
```

Since these changes are not written to disk, the context specified with this option does not persist between mounts. Therefore, this option must be used with the same context specified during every mount to retain the desired context. For information about making context mount persistent, refer to the Section 5.8.5, “Making Context Mounts Persistent”.

As an alternative to mounting file systems with **context** options, Booleans can be enabled to allow services access to file systems labeled with the **nfs_t** type. Refer to Managing Confined Services for instructions on configuring Booleans to allow services access to the **nfs_t** type.

5.8.4. Multiple NFS Mounts

When mounting multiple mounts from the same NFS export, attempting to override the SELinux context of each mount with a different context, results in subsequent mount commands failing. In the following example, the NFS server has a single export, **/export**, which has two subdirectories, **web/** and **database/**. The following commands attempt two mounts from a single NFS export, and try to override the context for each one:

```
~]# mount server:/export/web /local/web -o
context="system_u:object_r:httpd_sys_content_t:s0"

~]# mount server:/export/database /local/database -o
context="system_u:object_r:mysqld_db_t:s0"
```

The second mount command fails, and the following is logged to **/var/log/messages**:

```
kernel: SELinux: mount invalid.  Same superblock, different security
settings for (dev 0:15, type nfs)
```

To mount multiple mounts from a single NFS export, with each mount having a different context, use the **-o nosharecache,context** options. The following example mounts multiple mounts from a single NFS export, with a different context for each mount (allowing a single service access to each one):

```
~]# mount server:/export/web /local/web -o
nosharecache,context="system_u:object_r:httpd_sys_content_t:s0"

~]# mount server:/export/database /local/database -o \
nosharecache,context="system_u:object_r:mysqld_db_t:s0"
```

In this example, **server:/export/web** is mounted locally to **/local/web/**, with all files being labeled with the **httpd_sys_content_t** type, allowing Apache HTTP Server access. **server:/export/database** is mounted locally to **/local/database**, with all files being labeled with the **mysqld_db_t** type, allowing MySQL access. These type changes are not written to disk.

Important

The **nosharecache** options allows you to mount the same subdirectory of an export multiple times with different contexts (for example, mounting **/export/web** multiple times). Do not mount the same subdirectory from an export multiple times with different contexts, as this creates an overlapping mount, where files are accessible under two different contexts.

5.8.5. Making Context Mounts Persistent

To make context mounts persistent across remounting and reboots, add entries for the file systems in **/etc/fstab** or an automounter map, and use the desired context as a mount option. The following example adds an entry to **/etc/fstab** for an NFS context mount:

```
server:/export /local/mount/ nfs
context="system_u:object_r:httpd_sys_content_t:s0" 0 0
```

5.9. Maintaining SELinux Labels

These sections describe what happens to SELinux contexts when copying, moving, and archiving files and directories. Also, it explains how to preserve contexts when copying and archiving.

5.9.1. Copying Files and Directories

When a file or directory is copied, a new file or directory is created if it does not exist. That new file or directory's context is based on default-labeling rules, not the original file or directory's context (unless options were used to preserve the original context). For example, files created in user home directories are labeled with the **user_home_t** type:

```
~]$ touch file1
~]$ ls -Z file1
-rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
```

If such a file is copied to another directory, such as **/etc/**, the new file is created in accordance to default-labeling rules for the **/etc/** directory. Copying a file (without additional options) may not preserve the original context:

```
~]$ ls -Z file1
-rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
~]# cp file1 /etc/
~]$ ls -Z /etc/file1
-rw-r--r--  root root unconfined_u:object_r:etc_t:s0   /etc/file1
```

When **file1** is copied to **/etc/**, if **/etc/file1** does not exist, **/etc/file1** is created as a new file. As shown in the example above, **/etc/file1** is labeled with the **etc_t** type, in accordance to default-labeling rules.

When a file is copied over an existing file, the existing file's context is preserved, unless the user specified **cp** options to preserve the context of the original file, such as **--preserve=context**. SELinux policy may prevent contexts from being preserved during copies.

Copying Without Preserving SELinux Contexts

When copying a file with the **cp** command, if no options are given, the type is inherited from the targeted, parent directory:

```
~]$ touch file1
~]$ ls -Z file1
-rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
~]$ ls -dZ /var/www/html/
drwxr-xr-x  root root system_u:object_r:httpd_sys_content_t:s0
/var/www/html/
```

```
~]# cp file1 /var/www/html/
~]$ ls -Z /var/www/html/file1
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
/var/www/html/file1
```

In this example, **file1** is created in a user's home directory, and is labeled with the **user_home_t** type. The **/var/www/html/** directory is labeled with the **httpd_sys_content_t** type, as shown with the **ls -dZ /var/www/html/** command. When **file1** is copied to **/var/www/html/**, it inherits the **httpd_sys_content_t** type, as shown with the **ls -Z /var/www/html/file1** command.

Preserving SELinux Contexts When Copying

Use the **cp --preserve=context** command to preserve contexts when copying:

```
~]$ touch file1
~]$ ls -Z file1
-rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
~]$ ls -dZ /var/www/html/
drwxr-xr-x  root root system_u:object_r:httpd_sys_content_t:s0
/var/www/html/
~]# cp --preserve=context file1 /var/www/html/
~]$ ls -Z /var/www/html/file1
-rw-r--r--  root root unconfined_u:object_r:user_home_t:s0
/var/www/html/file1
```

In this example, **file1** is created in a user's home directory, and is labeled with the **user_home_t** type. The **/var/www/html/** directory is labeled with the **httpd_sys_content_t** type, as shown with the **ls -dZ /var/www/html/** command. Using the **--preserve=context** option preserves SELinux contexts during copy operations. As shown with the **ls -Z /var/www/html/file1** command, the **file1 user_home_t** type was preserved when the file was copied to **/var/www/html/**.

Copying and Changing the Context

Use the **cp -Z** command to change the destination copy's context. The following example was performed in the user's home directory:

```
~]$ touch file1
~]$ cp -Z system_u:object_r:samba_share_t:s0 file1 file2
~]$ ls -Z file1 file2
-rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
-rw-rw-r--  user1 group1 system_u:object_r:samba_share_t:s0 file2
~]$ rm file1 file2
```

In this example, the context is defined with the **-Z** option. Without the **-Z** option, **file2** would be labeled with the **unconfined_u:object_r:user_home_t** context.

Copying a File Over an Existing File

When a file is copied over an existing file, the existing file's context is preserved (unless an option is used to preserve contexts). For example:

```
~]# touch /etc/file1
```

```
~]# ls -Z /etc/file1
-rw-r--r--  root root unconfined_u:object_r:etc_t:s0   /etc/file1
~]# touch /tmp/file2
~]# ls -Z /tmp/file2
-rw-r--r--  root root unconfined_u:object_r:user_tmp_t:s0 /tmp/file2
~]# cp /tmp/file2 /etc/file1
~]# ls -Z /etc/file1
-rw-r--r--  root root unconfined_u:object_r:etc_t:s0   /etc/file1
```

In this example, two files are created: **/etc/file1**, labeled with the **etc_t** type, and **/tmp/file2**, labeled with the **user_tmp_t** type. The **cp /tmp/file2 /etc/file1** command overwrites **file1** with **file2**. After copying, the **ls -Z /etc/file1** command shows **file1** labeled with the **etc_t** type, not the **user_tmp_t** type from **/tmp/file2** that replaced **/etc/file1**.

Important

Copy files and directories, rather than moving them. This helps ensure they are labeled with the correct SELinux contexts. Incorrect SELinux contexts can prevent processes from accessing such files and directories.

5.9.2. Moving Files and Directories

Files and directories keep their current SELinux context when they are moved. In many cases, this is incorrect for the location they are being moved to. The following example demonstrates moving a file from a user's home directory to **/var/www/html/**, which is used by the Apache HTTP Server. Since the file is moved, it does not inherit the correct SELinux context:

1. Run the **cd** command without any arguments to change into your home directory. Once in your home directory, run the **touch file1** command to create a file. This file is labeled with the **user_home_t** type:

   ```
   ~]$ ls -Z file1
   -rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0 file1
   ```

2. Run the **ls -dZ /var/www/html/** command to view the SELinux context of the **/var/www/html/** directory:

   ```
   ~]$ ls -dZ /var/www/html/
   drwxr-xr-x  root root system_u:object_r:httpd_sys_content_t:s0
   /var/www/html/
   ```

   By default, the **/var/www/html/** directory is labeled with the **httpd_sys_content_t** type. Files and directories created under the **/var/www/html/** directory inherit this type, and as such, they are labeled with this type.

3. As the Linux root user, run the **mv file1 /var/www/html/** command to move **file1** to the **/var/www/html/** directory. Since this file is moved, it keeps its current **user_home_t** type:

   ```
   ~]# mv file1 /var/www/html/
   ~]# ls -Z /var/www/html/file1
   -rw-rw-r--  user1 group1 unconfined_u:object_r:user_home_t:s0
   /var/www/html/file1
   ```

By default, the Apache HTTP Server cannot read files that are labeled with the **user_home_t** type. If all files comprising a web page are labeled with the **user_home_t** type, or another type that the Apache HTTP Server cannot read, permission is denied when attempting to access them via web browsers, such as Firefox.

Important

Moving files and directories with the **mv** command may result in the incorrect SELinux context, preventing processes, such as the Apache HTTP Server and Samba, from accessing such files and directories.

5.9.3. Checking the Default SELinux Context

Use the **matchpathcon** command to check if files and directories have the correct SELinux context. From the matchpathcon(8) manual page: "**matchpathcon** queries the system policy and outputs the default security context associated with the file path." [11]. The following example demonstrates using the **matchpathcon** command to verify that files in **/var/www/html/** directory are labeled correctly:

1. As the Linux root user, run the **touch /var/www/html/file{1,2,3}** command to create three files (**file1**, **file2**, and **file3**). These files inherit the **httpd_sys_content_t** type from the **/var/www/html/** directory:

```
~]# touch /var/www/html/file{1,2,3}
~]# ls -Z /var/www/html/
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
file1
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
file2
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
file3
```

2. As the Linux root user, run the **chcon -t samba_share_t /var/www/html/file1** command to change the **file1** type to **samba_share_t**. Note that the Apache HTTP Server cannot read files or directories labeled with the **samba_share_t** type.

3. The **matchpathcon -V** option compares the current SELinux context to the correct, default context in SELinux policy. Run the **matchpathcon -V /var/www/html/*** command to check all files in the **/var/www/html/** directory:

```
~]$ matchpathcon -V /var/www/html/*
/var/www/html/file1 has context
unconfined_u:object_r:samba_share_t:s0, should be
system_u:object_r:httpd_sys_content_t:s0
/var/www/html/file2 verified.
/var/www/html/file3 verified.
```

The following output from the **matchpathcon** command explains that **file1** is labeled with the **samba_share_t** type, but should be labeled with the **httpd_sys_content_t** type:

```
/var/www/html/file1 has context unconfined_u:object_r:samba_share_t:s0,
should be system_u:object_r:httpd_sys_content_t:s0
```

To resolve the label problem and allow the Apache HTTP Server access to **file1**, as the Linux root user, run the **restorecon -v /var/www/html/file1** command:

```
~]# restorecon -v /var/www/html/file1
restorecon reset /var/www/html/file1 context
unconfined_u:object_r:samba_share_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
```

5.9.4. Archiving Files with tar

The **tar** utility does not retain extended attributes by default. Since SELinux contexts are stored in extended attributes, contexts can be lost when archiving files. Use the **tar --selinux** command to create archives that retain contexts. If a tar archive contains files without extended attributes, or if you want the extended attributes to match the system defaults, run the archive through the **restorecon** command:

```
~]$ tar -xvf archive.tar | restorecon -f -
```

Note that depending on the directory, you may need to be the Linux root user to run the **restorecon** command.

The following example demonstrates creating a tar archive that retains SELinux contexts:

1. As the Linux root user, run the **touch /var/www/html/file{1,2,3}** command to create three files (**file1**, **file2**, and **file3**). These files inherit the **httpd_sys_content_t** type from the **/var/www/html/** directory:

   ```
   ~]# touch /var/www/html/file{1,2,3}
   ~]# ls -Z /var/www/html/
   -rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
   file1
   -rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
   file2
   -rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
   file3
   ```

2. Run the **cd /var/www/html/** command to change into the **/var/www/html/** directory. Once in this directory, as the Linux root user, run the **tar --selinux -cf test.tar file{1,2,3}** command to create a tar archive named **test.tar**.

3. As the Linux root user, run the **mkdir /test** command to create a new directory, and then, run the **chmod 777 /test/** command to allow all users full-access to the **/test/** directory.

4. Run the **cp /var/www/html/test.tar /test/** command to copy the **test.tar** file in to the **/test/** directory.

5. Run the **cd /test/** command to change into the **/test/** directory. Once in this directory, run the **tar -xvf test.tar** command to extract the tar archive.

6. Run the **ls -lZ /test/** command to view the SELinux contexts. The **httpd_sys_content_t** type has been retained, rather than being changed to **default_t**, which would have happened had the **--selinux** not been used:

```
~]$ ls -lZ /test/
-rw-r--r--  user1 group1
unconfined_u:object_r:httpd_sys_content_t:s0 file1
-rw-r--r--  user1 group1
unconfined_u:object_r:httpd_sys_content_t:s0 file2
-rw-r--r--  user1 group1
unconfined_u:object_r:httpd_sys_content_t:s0 file3
-rw-r--r--  user1 group1 unconfined_u:object_r:default_t:s0
test.tar
```

7. If the **/test/** directory is no longer required, as the Linux root user, run the **rm -ri /test/** command to remove it, as well as all files in it.

Refer to the tar(1) manual page for further information about **tar**, such as the **--xattrs** option that retains all extended attributes.

5.9.5. Archiving Files with star

The **star** utility does not retain extended attributes by default. Since SELinux contexts are stored in extended attributes, contexts can be lost when archiving files. Use the **star -xattr -H=exustar** command to create archives that retain contexts. The *star* package is not installed by default. To install **star**, run the **yum install star** command as the Linux root user.

The following example demonstrates creating a Star archive that retains SELinux contexts:

1. As the Linux root user, run the **touch /var/www/html/file{1,2,3}** command to create three files (**file1**, **file2**, and **file3**). These files inherit the **httpd_sys_content_t** type from the **/var/www/html/** directory:

```
~]# touch /var/www/html/file{1,2,3}
~]# ls -Z /var/www/html/
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
file1
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
file2
-rw-r--r--  root root unconfined_u:object_r:httpd_sys_content_t:s0
file3
```

2. Run the **cd /var/www/html/** command to change into the **/var/www/html/** directory. Once in this directory, as the Linux root user, run the **star -xattr -H=exustar -c -f=test.star file{1,2,3}** command to create a Star archive named **test.star**:

```
~]# star -xattr -H=exustar -c -f=test.star file{1,2,3}
star: 1 blocks + 0 bytes (total of 10240 bytes = 10.00k).
```

3. As the Linux root user, run the **mkdir /test** command to create a new directory, and then, run the **chmod 777 /test/** command to allow all users full-access to the **/test/** directory.

4. Run the **cp /var/www/html/test.star /test/** command to copy the **test.star** file in to the **/test/** directory.

5. Run the **cd /test/** command to change into the **/test/** directory. Once in this directory, run the **star -x -f=test.star** command to extract the Star archive:

```
~]$ star -x -f=test.star
star: 1 blocks + 0 bytes (total of 10240 bytes = 10.00k).
```

6. Run the **ls -lZ /test/** command to view the SELinux contexts. The **httpd_sys_content_t** type has been retained, rather than being changed to **default_t**, which would have happened had the **-xattr -H=exustar** option not been used:

```
~]$ ls -lZ /test/
-rw-r--r--  user1 group1
unconfined_u:object_r:httpd_sys_content_t:s0 file1
-rw-r--r--  user1 group1
unconfined_u:object_r:httpd_sys_content_t:s0 file2
-rw-r--r--  user1 group1
unconfined_u:object_r:httpd_sys_content_t:s0 file3
-rw-r--r--  user1 group1 unconfined_u:object_r:default_t:s0
test.star
```

7. If the **/test/** directory is no longer required, as the Linux root user, run the **rm -ri /test/** command to remove it, as well as all files in it.

8. If **star** is no longer required, as the Linux root user, run the **yum remove star** command to remove the package.

Refer to the star(1) manual page for further information about **star**.

5.10. Information Gathering Tools

The utilities listed bellow are command-line tools that provide well-formatted information, such as access vector cache statistics or the number of classes, types, or Booleans.

avcstat

This command provides a short output of the access vector cache statistics since boot. You can watch the statistics in real time by specifying a time interval in seconds. This provides updated statistics since the initial output. The statistics file used is **/selinux/avc/cache_stats**, and you can specify a different cache file with the **-f /path/to/file** option.

```
~]# avcstat
   lookups       hits     misses     allocs   reclaims      frees
  47517410   47504630      12780      12780      12176      12275
```

seinfo

This utility is useful in describing the break-down of a policy, such as the number of classes, types, Booleans, allow rules, and others. **seinfo** is a command-line utility that uses a policy.conf file (a single text file containing policy source for versions 12 through 21), a binary policy file, a modular list of policy packages, or a policy list file as input. You must have the *setools-console* package installed to use the **seinfo** utility.

The output of **seinfo** will vary between binary and source files. For example, the policy source file uses the **{ }** brackets to group multiple rule elements onto a single line. A similar effect happens with attributes, where a single attribute expands into one or many types. Because these are expanded and no longer relevant in the binary policy file, they have a return value of zero in the search results.

However, the number of rules greatly increases as each formerly one line rule using brackets is now a number of individual lines.

Some items are not present in the binary policy. For example, neverallow rules are only checked during policy compile, not during runtime, and initial SIDs are not part of the binary policy since they are required prior to the policy being loaded by the kernel during boot.

```
~]# seinfo

Statistics for policy file: /etc/selinux/targeted/policy/policy.24
Policy Version  & Type: v.24 (binary, mls)

   Classes:            77    Permissions:       229
   Sensitivities:       1    Categories:       1024
   Types:            3001    Attributes:        244
   Users:               9    Roles:              13
   Booleans:          158    Cond. Expr.:       193
   Allow:          262796    Neverallow:          0
   Auditallow:         44    Dontaudit:      156710
   Type_trans:      10760    Type_change:        38
   Type_member:        44    Role allow:         20
   Role_trans:        237    Range_trans:      2546
   Constraints:        62    Validatetrans:       0
   Initial SIDs:       27    Fs_use:             22
   Genfscon:           82    Portcon:           373
   Netifcon:            0    Nodecon:             0
   Permissives:        22    Polcap:              2
```

The **seinfo** command can also list the number of types with the domain attribute, giving an estimate of the number of different confined processes:

```
~]# seinfo -adomain -x | wc -l
550
```

Not all domain types are confined. To look at the number of unconfined domains, use the unconfined_domain attribute:

```
~]# seinfo -aunconfined_domain_type -x | wc -l
52
```

Permissive domains can be counted with the **--permissive** option.

```
~]# seinfo --permissive -x | wc -l
31
```

Remove the **| wc -l** option in the above commands to see the full lists.

sesearch

You can use the **sesearch** command to search for a particular type in the policy. You can search either policy source files or the binary file. For example:

```
~]$ sesearch --role_allow -t httpd_sys_content_t
/etc/selinux/targeted/policy/policy.24
```

```
Found 20 role allow rules:
   allow system_r sysadm_r;
   allow sysadm_r system_r;
   allow sysadm_r staff_r;
   allow sysadm_r user_r;
   allow system_r git_shell_r;
   allow system_r guest_r;
   allow logadm_r system_r;
   allow system_r logadm_r;
   allow system_r nx_server_r;
   allow system_r staff_r;
   allow staff_r logadm_r;
   allow staff_r sysadm_r;
   allow staff_r unconfined_r;
   allow staff_r webadm_r;
   allow unconfined_r system_r;
   allow system_r unconfined_r;
   allow system_r user_r;
   allow webadm_r system_r;
   allow system_r webadm_r;
   allow system_r xguest_r;
```

The **sesearch** command can provide the number of *allow* rules:

```
~]# sesearch --allow | wc -l
262798
```

And the number of *dontaudit* rules:

```
~]# sesearch --dontaudit | wc -l
156712
```

5.11. Multi-Level Security (MLS)

The Multi-Level Security technology refers to a security scheme that enforces the Bell-La Padula Mandatory Access Model. Under MLS, users and processes are called *subjects*, and files, devices, and other passive components of the system are called *objects*. Both subjects and objects are labeled with a security level, which entails a subject's clearance or an object's classification. Each security level is composed of a *sensitivity* and a *category*, for example, an internal release schedule is filed under the internal documents category with a confidential sensitivity.

Figure 5.1, “Levels of clearance” shows levels of clearance as originally designed by the US defense community. Relating to our internal schedule example above, only users that have gained the confidential clearance are allowed to view documents in the confidential category. However, users who only have the confidential clearance are not allowed to view documents that require higher levels or clearance; they are allowed read access only to documents with lower levels of clearance, and write access to documents with higher levels of clearance.

Figure 5.1. Levels of clearance

Figure 5.2, “Allowed data flows using MLS” shows all allowed data flows between a subject running under the "Secret" security level and various objects with different security levels. In simple terms, the Bell-LaPadula model enforces two properties: *no read up* and *no write down*.

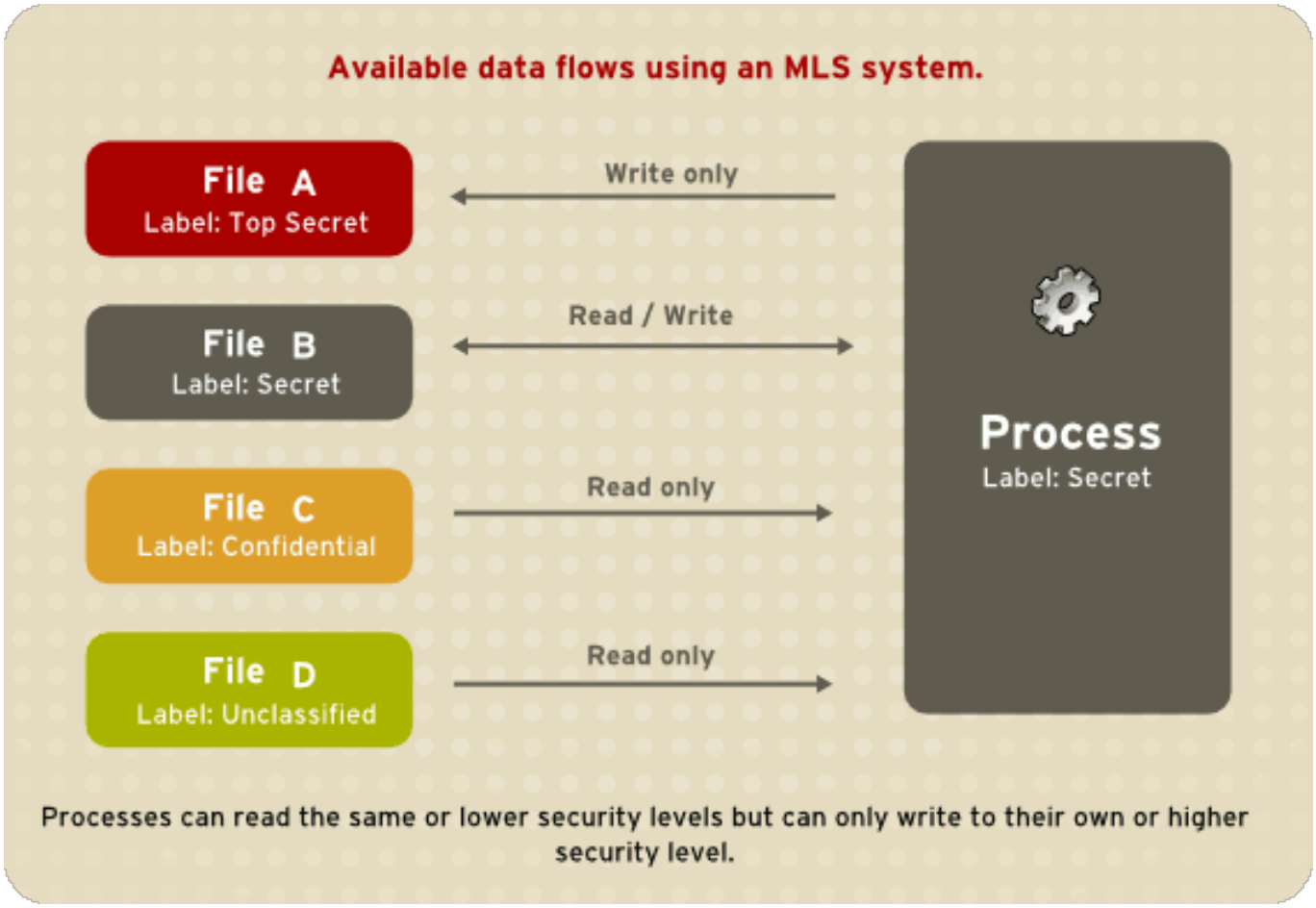


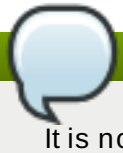
Figure 5.2. Allowed data flows using MLS

5.11.1. MLS and System Privileges

MLS access rules are always combined with conventional access permissions (file permissions). For example, if a user with a security level of "Secret" uses Discretionary Access Control (DAC) to block access to a file by other users, this also blocks access by users with a security level of "Top Secret". It is important to remember that SELinux MLS policy rules are checked *after* DAC rules. A higher security clearance does not automatically give permission to arbitrarily browse a file system.

Users with top-level clearances do not automatically acquire administrative rights on multi-level systems. While they may have access to all information on the computer, this is different from having administrative rights.

5.11.2. Enabling MLS in SELinux



Note

It is not recommended to use the MLS policy on a system that is running the X Window System.

Follow these steps to enable the SELinux MLS policy on your system.

1. Install the *selinux-policy-mls* package:

```
~]# yum install selinux-policy-mls
```

2. Before the MLS policy is enabled, each file on the file system must be relabeled with an MLS label. When the file system is relabeled, confined domains may be denied access, which may prevent your system from booting correctly. To prevent this from happening, configure **SELINUX=permissive** in the **/etc/selinux/config** file. Also, enable the MLS policy by configuring **SELINUXTYPE=mls**. Your configuration file should look like this:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=permissive
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       mls - Multi Level Security protection.
SELINUXTYPE=mls
```

3. Make sure SELinux is running in the permissive mode:

```
~]# setenforce 0
~]# getenforce
Permissive
```

4. Create the **.autorelabel** file in root's home directory to ensure that files are relabeled upon next reboot:

```
~]# touch /.autorelabel
```

5. Reboot your system. During the next boot, all file systems will be relabeled according to the MLS policy. The label process labels all files with an appropriate SELinux context:

```
*** Warning -- SELinux mls policy relabel is required.
*** Relabeling could take a very long time, depending on file
*** system size and speed of hard drives.
***********
```

   Each * (asterisk) character on the bottom line represents 1000 files that have been labeled. In the above example, eleven * characters represent 11000 files which have been labeled. The time it takes to label all files depends upon the number of files on the system, and the speed of the hard disk drives. On modern systems, this process can take as little as 10 minutes. Once the labeling process finishes, the system will automatically reboot.

6. In permissive mode, SELinux policy is not enforced, but denials are still logged for actions that would have been denied if running in enforcing mode. Before changing to enforcing mode, as the Linux root user, run the **grep "SELinux is preventing" /var/log/messages** command to confirm that SELinux did not deny actions during the last boot. If SELinux did not deny actions during the last boot, this command does not return any output. Refer to Chapter 8, *Troubleshooting* for troubleshooting information if SELinux denied access during boot.

7. If there were no denial messages in **/var/log/messages**, or you have resolved all existing denials, configure **SELINUX=enforcing** in the **/etc/selinux/config** file:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=enforcing
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       mls - Multi Level Security protection.
SELINUXTYPE=mls
```

8. Reboot your system and make sure SELinux is running in permissive mode:

```
~]$ getenforce
Enforcing
```

   and the MLS policy is enabled:

```
~]# sestatus |grep mls
Policy from config file:        mls
```

5.11.3. Creating a User With a Specific MLS Range

Follow these steps to create a new Linux user with a specific MLS range:

1. Add a new Linux user via the **useradd** command and map the new Linux user to an existing SELinux user (in this case, **user_u**):

```
~]# useradd -Z user_u john
```

2. Assign the newly-created Linux user a password:

```
~]# passwd john
```

3. Run the **semanage login -l** command to view the mapping between SELinux and Linux users. The output should be as follows:

```
Login Name                SELinux User              MLS/MCS Range

__default__               user_u                    s0
john                      user_u                    s0
root                      root                      s0-s15:c0.c1023
system_u                  system_u                  s0-s15:c0.c1023
```

4. Define a specific range for user **john**:

```
~]# semanage login --modify --seuser user_u --range s2:c100 john
```

5. Run the **semanage login -l** command to view the mapping between SELinux and Linux users. Note that the user **john** now has a specific MLS range defined:

```
Login Name                SELinux User              MLS/MCS Range

__default__               user_u                    s0
john                      user_u                    s2:c100
root                      root                      s0-s15:c0.c1023
system_u                  system_u                  s0-s15:c0.c1023
```

6. To correct the label on john's home directory (if needed), run the following command:

```
~]# chcon -R -l s2:c100 /home/john
```

5.11.4. Setting Up Polyinstantiated Directories

The **/tmp/** and **/var/tmp/** directories are normally used for temporary storage by all programs, services, and users. Such setup, however, makes these directories vulnerable to race condition attacks, or an information leak based on file names. SELinux offers a solution in the form of polyinstantiated directories. This effectively means that both **/tmp/** and **/var/tmp/** are instantiated, making them appear private for each user. When instantiation of directories is enabled, each user's **/tmp/** and **/var/tmp/** directory is automatically mounted under **/tmp-inst** and **/var/tmp/tmp-inst**.

Follow these steps to enable polyinstantiation of directories:

1. Uncomment the last three lines in the **/etc/security/namespace.conf** file to enable instantiation of the **/tmp/**, **/var/tmp/**, and users' home directories:

```
~]$ tail -n 3 /etc/security/namespace.conf
/tmp     /tmp-inst/            level      root,adm
/var/tmp /var/tmp/tmp-inst/    level      root,adm
$HOME    $HOME/$USER.inst/     level
```

2. Ensure that in the **/etc/pam.d/login** file, the **pam_namespace.so** module is configured for session:

```
~]$ grep namespace /etc/pam.d/login
session    required     pam_namespace.so
```

3. Reboot your system.

---

[7] Brindle, Joshua. "Re: blurb for fedora setools packages" Email to Murray McAllister. 1 November 2008. Any edits or changes in this version were done by Murray McAllister.

[8] To temporarily revert to the default behavior, as the Linux root user, run the **setsebool httpd_can_network_connect_db off** command. For changes that persist across reboots, run the **setsebool -P httpd_can_network_connect_db off** command.

[9] Files in the **/etc/selinux/targeted/contexts/files/** directory define contexts for files and directories. Files in this directory are read by the **restorecon** and **setfiles** utilities to restore files and directories to their default contexts.

[10] Morris, James. "Filesystem Labeling in SELinux". Published 1 October 2004. Accessed 14 October 2008: http://www.linuxjournal.com/article/7426.

[11] The matchpathcon(8) manual page, as shipped with the *libselinux-utils* package in Red Hat Enterprise Linux, is written by Daniel Walsh. Any edits or changes in this version were done by Murray McAllister.

Chapter 6. Confining Users

A number of confined SELinux users are available in Red Hat Enterprise Linux 6. Each Linux user is mapped to an SELinux user via SELinux policy, allowing Linux users to inherit the restrictions placed on SELinux users, for example (depending on the user), not being able to: run the X Window System; use networking; run setuid applications (unless SELinux policy permits it); or run the **su** and **sudo** commands. This helps protect the system from the user. Refer to Section 4.3, "Confined and Unconfined Users" for further information about confined users.

6.1. Linux and SELinux User Mappings

As the Linux root user, run the **semanage login -l** command to view the mapping between Linux users and SELinux users:

```
~]# semanage login -l

Login Name                SELinux User              MLS/MCS Range

__default__               unconfined_u              s0-s0:c0.c1023
root                      unconfined_u              s0-s0:c0.c1023
system_u                  system_u                  s0-s0:c0.c1023
```

In Red Hat Enterprise Linux 6, Linux users are mapped to the SELinux **__default__** login by default (which is in turn mapped to the SELinux **unconfined_u** user). When a Linux user is created with the **useradd** command, if no options are specified, they are mapped to the SELinux **unconfined_u** user. The following defines the default-mapping:

```
__default__               unconfined_u              s0-s0:c0.c1023
```

6.2. Confining New Linux Users: useradd

Linux users mapped to the SELinux **unconfined_u** user run in the **unconfined_t** domain. This is seen by running the **id -Z** command while logged-in as a Linux user mapped to **unconfined_u**:

```
~]$ id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

When Linux users run in the **unconfined_t** domain, SELinux policy rules are applied, but policy rules exist that allow Linux users running in the **unconfined_t** domain almost all access. If unconfined Linux users execute an application that SELinux policy defines can transition from the **unconfined_t** domain to its own confined domain, unconfined Linux users are still subject to the restrictions of that confined domain. The security benefit of this is that, even though a Linux user is running unconfined, the application remains confined, and therefore, the exploitation of a flaw in the application can be limited by policy.

Note

This does not protect the system from the user. Instead, the user and the system are being protected from possible damage caused by a flaw in the application.

When creating Linux users with the **useradd** command, use the **-Z** option to specify which SELinux user they are mapped to. The following example creates a new Linux user, **useruuser**, and maps that user to the SELinux **user_u** user. Linux users mapped to the SELinux **user_u** user run in the **user_t** domain. In this domain, Linux users are unable to run setuid applications unless SELinux policy permits it (such as **passwd**), and cannot run the **su** or **sudo** command, preventing them from becoming the Linux root user with these commands.

1. As the Linux root user, run the **useradd -Z user_u useruuser** command to create a new Linux user (**useruuser**) that is mapped to the SELinux **user_u** user.

2. As the Linux root user, run the **semanage login -l** command to view the mapping between the Linux **useruuser** user and **user_u**:

```
~]# semanage login -l

Login Name                SELinux User              MLS/MCS Range

__default__               unconfined_u              s0-s0:c0.c1023
root                      unconfined_u              s0-s0:c0.c1023
system_u                  system_u                  s0-s0:c0.c1023
useruuser                 user_u                    s0
```

3. As the Linux root user, run the **passwd useruuser** command to assign a password to the Linux **useruuser** user:

```
~]# passwd useruuser
Changing password for user useruuser.
New UNIX password: Enter a password
Retype new UNIX password: Enter the same password again
passwd: all authentication tokens updated successfully.
```

4. Log out of your current session, and log in as the Linux **useruuser** user. When you log in, pam_selinux maps the Linux user to an SELinux user (in this case, **user_u**), and sets up the resulting SELinux context. The Linux user's shell is then launched with this context. Run the **id -Z** command to view the context of a Linux user:

```
~]$ id -Z
user_u:user_r:user_t:s0
```

5. Log out of the Linux **useruuser**'s session, and log back in with your account. If you do not want the Linux **useruuser** user, run the **userdel -r useruuser** command as the Linux root user to remove it, along with its home directory.

6.3. Confining Existing Linux Users: semanage login

If a Linux user is mapped to the SELinux **unconfined_u** user (the default behavior), and you would like to change which SELinux user they are mapped to, use the **semanage login** command. The following example creates a new Linux user named **newuser**, then maps that Linux user to the SELinux **user_u** user:

1. As the Linux root user, run the **useradd newuser** command to create a new Linux user (**newuser**). Since this user uses the default mapping, it does not appear in the **semanage login -l** output:

```
~]# useradd newuser
~]# semanage login -l

Login Name                 SELinux User               MLS/MCS Range

__default__                unconfined_u               s0-s0:c0.c1023
root                       unconfined_u               s0-s0:c0.c1023
system_u                   system_u                   s0-s0:c0.c1023
```

2. To map the Linux **newuser** user to the SELinux **user_u** user, run the following command as the Linux root user:

```
~]# semanage login -a -s user_u newuser
```

The **-a** option adds a new record, and the **-s** option specifies the SELinux user to map a Linux user to. The last argument, **newuser**, is the Linux user you want mapped to the specified SELinux user.

3. To view the mapping between the Linux **newuser** user and **user_u**, run the **semanage login -l** command as the Linux root user:

```
~]# semanage login -l

Login Name                 SELinux User               MLS/MCS Range

__default__                unconfined_u               s0-s0:c0.c1023
newuser                    user_u                     s0
root                       unconfined_u               s0-s0:c0.c1023
system_u                   system_u                   s0-s0:c0.c1023
```

4. As the Linux root user, run the **passwd newuser** command to assign a password to the Linux **newuser** user:

```
~]# passwd newuser
Changing password for user newuser.
New password: Enter a password
Retype new password: Enter the same password again
passwd: all authentication tokens updated successfully.
```

5. Log out of your current session, and log in as the Linux **newuser** user. Run the **id -Z** command to view the **newuser**'s SELinux context:

```
~]$ id -Z
user_u:user_r:user_t:s0
```

6. Log out of the Linux **newuser**'s session, and log back in with your account. If you do not want the Linux **newuser** user, run the **userdel -r newuser** command as the Linux root user to remove it, along with its home directory. Run the **semanage login -d newuser** command to remove the mapping between the Linux **newuser** user and **user_u**:

```
~]# userdel -r newuser
~]# semanage login -d newuser
~]# semanage login -l
```

```
Login Name                SELinux User                MLS/MCS Range

__default__               unconfined_u                s0-s0:c0.c1023
root                      unconfined_u                s0-s0:c0.c1023
system_u                  system_u                    s0-s0:c0.c1023
```

6.4. Changing the Default Mapping

In Red Hat Enterprise Linux 6, Linux users are mapped to the SELinux **__default__** login by default (which is in turn mapped to the SELinux **unconfined_u** user). If you would like new Linux users, and Linux users not specifically mapped to an SELinux user to be confined by default, change the default mapping with the **semanage login** command.

For example, run the following command as the Linux root user to change the default mapping from **unconfined_u** to **user_u**:

```
~]# semanage login -m -S targeted -s "user_u" -r s0 __default__
```

Run the **semanage login -l** command as the Linux root user to verify the **__default__** login is mapped to **user_u**:

```
~]# semanage login -l

Login Name                SELinux User                MLS/MCS Range

__default__               user_u                      s0
root                      unconfined_u                s0-s0:c0.c1023
system_u                  system_u                    s0-s0:c0.c1023
```

If a new Linux user is created and an SELinux user is not specified, or if an existing Linux user logs in and does not match a specific entry from the **semanage login -l** output, they are mapped to **user_u**, as per the **__default__** login.

To change back to the default behavior, run the following command as the Linux root user to map the **__default__** login to the SELinux **unconfined_u** user:

```
~]# semanage login -m -S targeted -s "unconfined_u" -r s0-s0:c0.c1023
__default__
```

6.5. xguest: Kiosk Mode

The *xguest* package provides a kiosk user account. This account is used to secure machines that people walk up to and use, such as those at libraries, banks, airports, information kiosks, and coffee shops. The kiosk user account is very limited: essentially, it only allows users to log in and use **Firefox** to browse Internet websites. Any changes made while logged in with this account, such as creating files or changing settings, are lost when you log out.

To set up the kiosk account:

1. As the Linux root user, run the **yum install xguest** command to install the *xguest* package. Install dependencies as required.

2. In order to allow the kiosk account to be used by a variety of people, the account is not password-protected, and as such, the account can only be protected if SELinux is running in enforcing mode. Before logging in with this account, use the **getenforce** command to confirm that SELinux is running in enforcing mode:

```
~]$ getenforce
Enforcing
```

   If this is not the case, refer to Section 2.4, "SELinux Modes" for information about changing to enforcing mode. It is not possible to log in with this account if SELinux is in permissive mode or disabled.

3. You can only log in to this account via the GNOME Display Manager (GDM). Once the *xguest* package is installed, a **Guest** account is added to the GDM login screen.

6.6. Booleans for Users Executing Applications

Not allowing Linux users to execute applications (which inherit users' permissions) in their home directories and **/tmp/**, which they have write access to, helps prevent flawed or malicious applications from modifying files that users own. In Red Hat Enterprise Linux 6, by default, Linux users in the **guest_t** and **xguest_t** domains cannot execute applications in their home directories or **/tmp/**; however, by default, Linux users in the **user_t** and **staff_t** domains can.

Booleans are available to change this behavior, and are configured with the **setsebool** command. The **setsebool** command must be run as the Linux root user. The **setsebool -P** command makes persistent changes. Do not use the **-P** option if you do not want changes to persist across reboots:

guest_t

To *allow* Linux users in the **guest_t** domain to execute applications in their home directories and **/tmp/**:

```
~]# setsebool -P allow_guest_exec_content on
```

xguest_t

To *allow* Linux users in the **xguest_t** domain to execute applications in their home directories and **/tmp/**:

```
~]# setsebool -P allow_xguest_exec_content on
```

user_t

To *prevent* Linux users in the **user_t** domain from executing applications in their home directories and **/tmp/**:

```
~]# setsebool -P allow_user_exec_content off
```

staff_t

To *prevent* Linux users in the **staff_t** domain from executing applications in their home directories and **/tmp/**:

```
~]# setsebool -P allow_staff_exec_content off
```

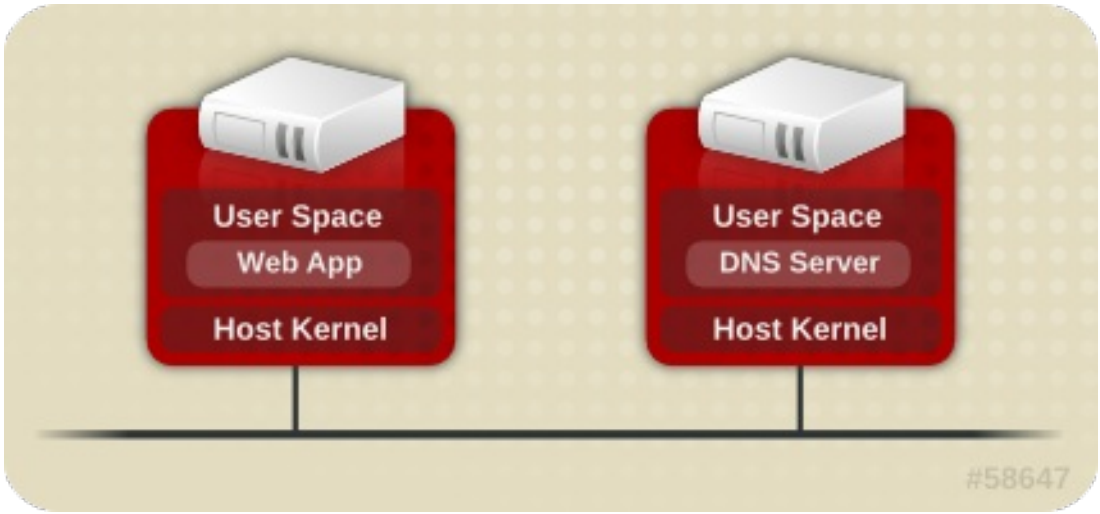
Chapter 7. sVirt

sVirt is a technology included in Red Hat Enterprise Linux 6 that integrates SELinux and virtualization. sVirt applies Mandatory Access Control (MAC) to improve security when using virtual machines. The main reasons for integrating these technologies are to improve security and harden the system against bugs in the hypervisor that might be used as an attack vector aimed toward the host or to another virtual machine.

This chapter describes how sVirt integrates with virtualization technologies in Red Hat Enterprise Linux 6.

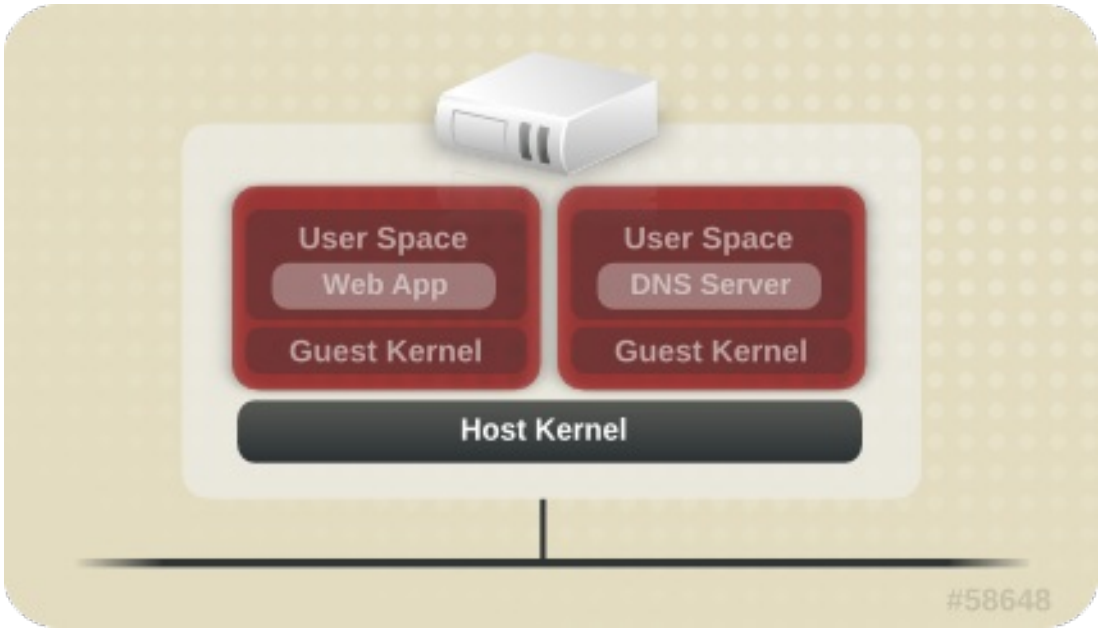
Non-Virtualized Environment

In a non-virtualized environment, hosts are separated from each other physically and each host has a self-contained environment, consisting of services such as a Web server, or a DNS server. These services communicate directly to their own user space, host kernel and physical host, offering their services directly to the network. The following image represents a non-virtualized environment:



Virtualized Environment

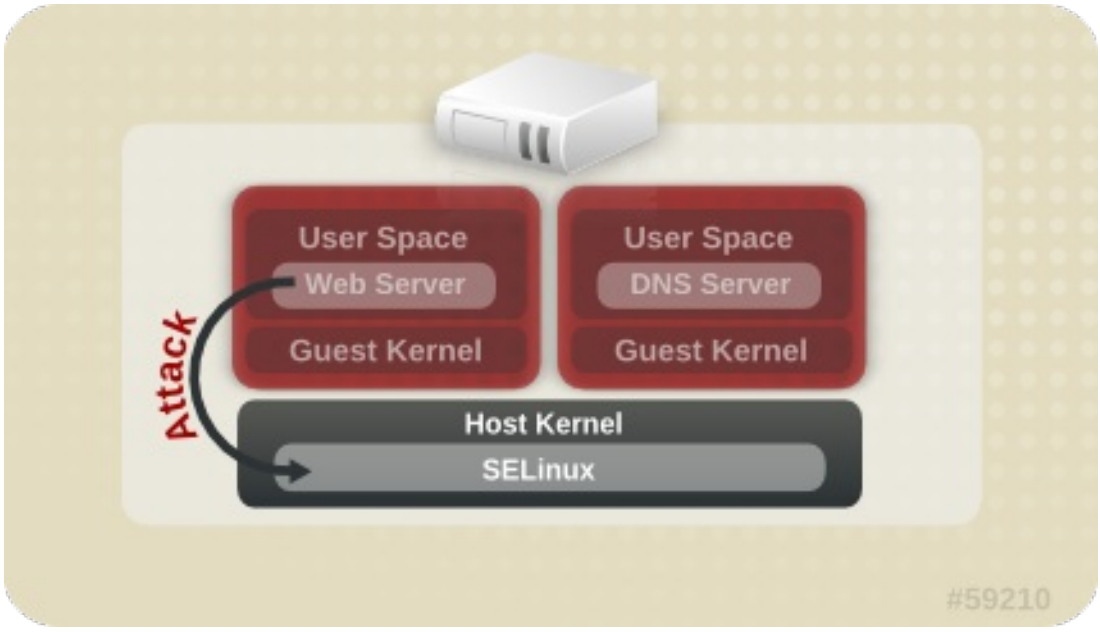
In a virtualized environment, several operating systems can be housed (as "guests") within a single host kernel and physical host. The following image represents a virtualized environment:



7.1. Security and Virtualization

When services are not virtualized, machines are physically separated. Any exploit is usually contained to the affected machine, with the obvious exception of network attacks. When services are grouped together in a virtualized environment, extra vulnerabilities emerge in the system. If there is a security flaw in the hypervisor that can be exploited by a guest instance, this guest may be able to not only attack the host, but also other guests running on that host. This is not theoretical; attacks already exist on hypervisors. These attacks can extend beyond the guest instance and could expose other guests to attack.

sVirt is an effort to isolate guests and limit their ability to launch further attacks if exploited. This is demonstrated in the following image, where an attack cannot break out of the virtual machine and extend to another host instance:



SELinux introduces a pluggable security framework for virtualized instances in its implementation of Mandatory Access Control (MAC). The sVirt framework allows guests and their resources to be uniquely labeled. Once labeled, rules can be applied which can reject access between different guests.

7.2. sVirt Labeling

Like other services under the protection of SELinux, sVirt uses process-based mechanisms and restrictions to provide an extra layer of security over guest instances. Under typical use, you should not even notice that sVirt is working in the background. This section describes the labeling features of sVirt.

As shown in the following output, when using sVirt, each Virtual Machine (VM) process is labeled and runs with a dynamically generated level. Each process is isolated from other VMs with different levels:

```
~]# ps -eZ | grep qemu

system_u:system_r:svirt_t:s0:c87,c520 27950 ?  00:00:17 qemu-kvm
system_u:system_r:svirt_t:s0:c639,c757 27989 ? 00:00:06 qemu-system-x86
```

The actual disk images are automatically labeled to match the processes, as shown in the following output:

```
~]# ls -lZ /var/lib/libvirt/images/*

system_u:object_r:svirt_image_t:s0:c87,c520   image1
```

The following table outlines the different labels that can be assigned when using sVirt:

Table 7.1. sVirt Labels

Type	SELinux Context	Description
Virtual Machine Processes	system_u:system_r:svirt_t:MCS1	MCS1 is a randomly selected MCS field. Currently approximately 500,000 labels are supported.
Virtual Machine Image	system_u:object_r:svirt_image_t :MCS1	Only processes labeled *svirt_t* with the same MCS fields are able to read/write these image files and devices.
Virtual Machine Shared Read/Write Content	system_u:object_r:svirt_image_t :s0	All processes labeled *svirt_t* are allowed to write to the svirt_image_t:s0 files and devices.
Virtual Machine Image	system_u:object_r:virt_content_t :s0	System default label used when an image exits. No *svirt_t* virtual processes are allowed to read files/devices with this label.

It is also possible to perform static labeling when using sVirt. Static labels allow the administrator to select a specific label, including the MCS/MLS field, for a virtual machine. Administrators who run statically-labeled virtual machines are responsible for setting the correct label on the image files. The virtual machine will always be started with that label, and the sVirt system will never modify the label of a statically-labeled virtual machine's content. This allows the sVirt component to run in an MLS environment. You can also run multiple virtual machines with different sensitivity levels on a system, depending on your requirements.

Chapter 8. Troubleshooting

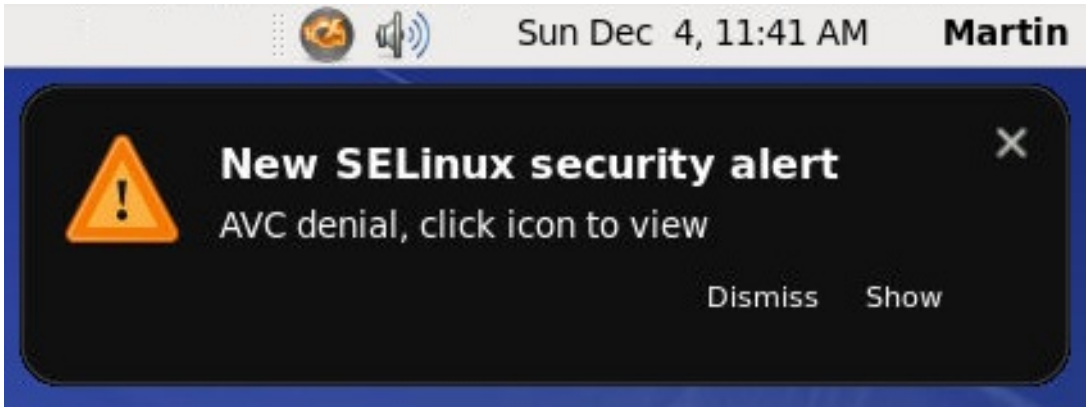
The following chapter describes what happens when SELinux denies access; the top three causes of problems; where to find information about correct labeling; analyzing SELinux denials; and creating custom policy modules with **audit2allow**.

8.1. What Happens when Access is Denied

SELinux decisions, such as allowing or disallowing access, are cached. This cache is known as the Access Vector Cache (AVC). Denial messages are logged when SELinux denies access. These denials are also known as "AVC denials", and are logged to a different location, depending on which daemons are running:

Daemon	Log Location
auditd on	**/var/log/audit/audit.log**
auditd off; rsyslogd on	**/var/log/messages**
setroubleshootd, rsyslogd, and auditd on	**/var/log/audit/audit.log**. Easier-to-read denial messages also sent to **/var/log/messages**

If you are running the X Window System, have the *setroubleshoot* and *setroubleshoot-server* packages installed, and the **setroubleshootd** and **auditd** daemons are running, a warning is displayed when access is denied by SELinux:



Clicking on 'Show' presents a detailed analysis of why SELinux denied access, and a possible solution for allowing access. If you are not running the X Window System, it is less obvious when access is denied by SELinux. For example, users browsing your website may receive an error similar to the following:

```
Forbidden

You don't have permission to access file name on this server
```

For these situations, if DAC rules (standard Linux permissions) allow access, check **/var/log/messages** and **/var/log/audit/audit.log** for **"SELinux is preventing"** and **"denied"** errors respectively. This can be done by running the following commands as the Linux root user:

```
~]# grep "SELinux is preventing" /var/log/messages
```

```
~]# grep "denied" /var/log/audit/audit.log
```

8.2. Top Three Causes of Problems

The following sections describe the top three causes of problems: labeling problems, configuring Booleans and ports for services, and evolving SELinux rules.

8.2.1. Labeling Problems

On systems running SELinux, all processes and files are labeled with a label that contains security-relevant information. This information is called the SELinux context. If these labels are wrong, access may be denied. If an application is labeled incorrectly, the process it transitions to may not have the correct label, possibly causing SELinux to deny access, and the process being able to create mislabeled files.

A common cause of labeling problems is when a non-standard directory is used for a service. For example, instead of using **/var/www/html/** for a website, an administrator wants to use **/srv/myweb/**. On Red Hat Enterprise Linux 6, the **/srv/** directory is labeled with the **var_t** type. Files and directories created and **/srv/** inherit this type. Also, newly-created top-level directories (such as **/myserver/**) may be labeled with the **default_t** type. SELinux prevents the Apache HTTP Server (**httpd**) from accessing both of these types. To allow access, SELinux must know that the files in **/srv/myweb/** are to be accessible to **httpd**:

```
~]# semanage fcontext -a -t httpd_sys_content_t "/srv/myweb(/.*)?"
```

This **semanage** command adds the context for the **/srv/myweb/** directory (and all files and directories under it) to the SELinux file-context configuration [12]. The **semanage** command does not change the context. As the Linux root user, run the **restorecon** command to apply the changes:

```
~]# restorecon -R -v /srv/myweb
```

Refer to Section 5.6.2, “Persistent Changes: semanage fcontext” for further information about adding contexts to the file-context configuration.

8.2.1.1. What is the Correct Context?

The **matchpathcon** command checks the context of a file path and compares it to the default label for that path. The following example demonstrates using **matchpathcon** on a directory that contains incorrectly labeled files:

```
~]$ matchpathcon -V /var/www/html/*
/var/www/html/index.html has context
unconfined_u:object_r:user_home_t:s0, should be
system_u:object_r:httpd_sys_content_t:s0
/var/www/html/page1.html has context
unconfined_u:object_r:user_home_t:s0, should be
system_u:object_r:httpd_sys_content_t:s0
```

In this example, the **index.html** and **page1.html** files are labeled with the **user_home_t** type. This type is used for files in user home directories. Using the **mv** command to move files from your home directory may result in files being labeled with the **user_home_t** type. This type should not exist outside of home directories. Use the **restorecon** command to restore such files to their correct type:

```
~]# restorecon -v /var/www/html/index.html
restorecon reset /var/www/html/index.html context
unconfined_u:object_r:user_home_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
```

To restore the context for all files under a directory, use the **-R** option:

```
~]# restorecon -R -v /var/www/html/
restorecon reset /var/www/html/page1.html context
unconfined_u:object_r:samba_share_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
restorecon reset /var/www/html/index.html context
unconfined_u:object_r:samba_share_t:s0-
>system_u:object_r:httpd_sys_content_t:s0
```

Refer to Section 5.9.3, “Checking the Default SELinux Context” for a more detailed example of **matchpathcon**.

8.2.2. How are Confined Services Running?

Services can be run in a variety of ways. To cater for this, you must tell SELinux how you are running services. This can be achieved via Booleans that allow parts of SELinux policy to be changed at runtime, without any knowledge of SELinux policy writing. This allows changes, such as allowing services access to NFS volumes, without reloading or recompiling SELinux policy. Also, running services on non-default port numbers requires policy configuration to be updated via the **semanage** command.

For example, to allow the Apache HTTP Server to communicate with MySQL, enable the **httpd_can_network_connect_db** Boolean:

```
~]# setsebool -P httpd_can_network_connect_db on
```

If access is denied for a particular service, use the **getsebool** and **grep** commands to see if any Booleans are available to allow access. For example, use the **getsebool -a | grep ftp** command to search for FTP related Booleans:

```
~]$ getsebool -a | grep ftp
allow_ftpd_anon_write --> off
allow_ftpd_full_access --> off
allow_ftpd_use_cifs --> off
allow_ftpd_use_nfs --> off
ftp_home_dir --> off
ftpd_connect_db --> off
httpd_enable_ftp_server --> off
tftp_anon_write --> off
```

For a list of Booleans and whether they are on or off, run the **getsebool -a** command. For a list of Booleans, an explanation of what each one is, and whether they are on or off, run the **semanage boolean -l** command as the Linux root user. Refer to Section 5.5, “Booleans” for information about listing and configuring Booleans.

Port Numbers

Depending on policy configuration, services may only be allowed to run on certain port numbers. Attempting to change the port a service runs on without changing policy may result in the service failing to start. For example, run the **semanage port -l | grep http** command as the Linux root user to list **http** related ports:

```
~]# semanage port -l | grep http
http_cache_port_t              tcp      3128, 8080, 8118
http_cache_port_t              udp      3130
http_port_t                    tcp      80, 443, 488, 8008, 8009, 8443
pegasus_http_port_t            tcp      5988
pegasus_https_port_t           tcp      5989
```

The **http_port_t** port type defines the ports Apache HTTP Server can listen on, which in this case, are TCP ports 80, 443, 488, 8008, 8009, and 8443. If an administrator configures **httpd.conf** so that **httpd** listens on port 9876 (**Listen 9876**), but policy is not updated to reflect this, the **service httpd start** command fails:

```
~]# service httpd start
Starting httpd: (13)Permission denied: make_sock: could not bind to
address [::]:9876
(13)Permission denied: make_sock: could not bind to address 0.0.0.0:9876
no listening sockets available, shutting down
Unable to open logs
                   [FAILED]
```

An SELinux denial similar to the following is logged to **/var/log/audit/audit.log**:

```
type=AVC msg=audit(1225948455.061:294): avc:  denied  { name_bind } for
pid=4997 comm="httpd" src=9876 scontext=unconfined_u:system_r:httpd_t:s0
tcontext=system_u:object_r:port_t:s0 tclass=tcp_socket
```

To allow **httpd** to listen on a port that is not listed for the **http_port_t** port type, run the **semanage port** command to add a port to policy configuration [13]:

```
~]# semanage port -a -t http_port_t -p tcp 9876
```

The **-a** option adds a new record; the **-t** option defines a type; and the **-p** option defines a protocol. The last argument is the port number to add.

8.2.3. Evolving Rules and Broken Applications

Applications may be broken, causing SELinux to deny access. Also, SELinux rules are evolving – SELinux may not have seen an application running in a certain way, possibly causing it to deny access, even though the application is working as expected. For example, if a new version of PostgreSQL is released, it may perform actions the current policy has not seen before, causing access to be denied, even though access should be allowed.

For these situations, after access is denied, use **audit2allow** to create a custom policy module to allow access. Refer to Section 8.3.8, “Allowing Access: audit2allow” for information about using **audit2allow**.

8.3. Fixing Problems

The following sections help troubleshoot issues. They go over: checking Linux permissions, which are checked before SELinux rules; possible causes of SELinux denying access, but no denials being logged; manual pages for services, which contain information about labeling and Booleans; permissive domains, for allowing one process to run permissive, rather than the whole system; how to search for and view denial messages; analyzing denials; and creating custom policy modules with **audit2allow**.

8.3.1. Linux Permissions

When access is denied, check standard Linux permissions. As mentioned in Chapter 2, *Introduction*, most operating systems use a Discretionary Access Control (DAC) system to control access, allowing users to control the permissions of files that they own. SELinux policy rules are checked after DAC rules. SELinux policy rules are not used if DAC rules deny access first.

If access is denied and no SELinux denials are logged, use the **ls -l** command to view the standard Linux permissions:

```
~]$ ls -l /var/www/html/index.html
-rw-r----- 1 root root 0 2009-05-07 11:06 index.html
```

In this example, **index.html** is owned by the root user and group. The root user has read and write permissions (**-rw**), and members of the root group have read permissions (**-r-**). Everyone else has no access (**---**). By default, such permissions do not allow **httpd** to read this file. To resolve this issue, use the **chown** command to change the owner and group. This command must be run as the Linux root user:

```
~]# chown apache:apache /var/www/html/index.html
```

This assumes the default configuration, in which **httpd** runs as the Linux apache user. If you run **httpd** with a different user, replace **apache:apache** with that user.

Refer to the Fedora Documentation Project "Permissions" draft for information about managing Linux permissions.

8.3.2. Possible Causes of Silent Denials

In certain situations, AVC denials may not be logged when SELinux denies access. Applications and system library functions often probe for more access than required to perform their tasks. To maintain least privilege without filling audit logs with AVC denials for harmless application probing, the policy can silence AVC denials without allowing a permission by using **dontaudit** rules. These rules are common in standard policy. The downside of **dontaudit** is that, although SELinux denies access, denial messages are not logged, making troubleshooting more difficult.

To temporarily disable **dontaudit** rules, allowing all denials to be logged, run the following command as the Linux root user:

```
~]# semodule -DB
```

The **-D** option disables **dontaudit** rules; the **-B** option rebuilds policy. After running **semodule -DB**, try exercising the application that was encountering permission problems, and see if SELinux denials — relevant to the application — are now being logged. Take care in deciding which denials should be allowed, as some should be ignored and handled via **dontaudit** rules. If in doubt, or in search of guidance, contact other SELinux users and developers on an SELinux list, such as fedora-selinux-list.

To rebuild policy and enable **dontaudit** rules, run the following command as the Linux root user:

```
~]# semodule -B
```

This restores the policy to its original state. For a full list of **dontaudit** rules, run the **sesearch --dontaudit** command. Narrow down searches using the **-s *domain*** option and the **grep** command. For example:

```
~]$ sesearch --dontaudit -s smbd_t | grep squid
dontaudit smbd_t squid_port_t : tcp_socket name_bind ;
dontaudit smbd_t squid_port_t : udp_socket name_bind ;
```

Refer to Section 8.3.6, “Raw Audit Messages” and Section 8.3.7, “sealert Messages” for information about analyzing denials.

8.3.3. Manual Pages for Services

Manual pages for services contain valuable information, such as what file type to use for a given situation, and Booleans to change the access a service has (such as **httpd** accessing NFS volumes). This information may be in the standard manual page, or a manual page with **selinux** prepended or appended.

For example, the httpd_selinux(8) manual page has information about what file type to use for a given situation, as well as Booleans to allow scripts, sharing files, accessing directories inside user home directories, and so on. Other manual pages with SELinux information for services include:

- Samba: the samba_selinux(8) manual page describes that files and directories to be exported via Samba must be labeled with the **samba_share_t** type, as well as Booleans to allow files labeled with types other than **samba_share_t** to be exported via Samba.
- Berkeley Internet Name Domain (BIND): the named(8) manual page describes what file type to use for a given situation (see the **Red Hat SELinux BIND Security Profile** section). The named_selinux(8) manual page describes that, by default, **named** cannot write to master zone files, and to allow such access, the **named_write_master_zones** Boolean must be enabled.

The information in manual pages helps you configure the correct file types and Booleans, helping to prevent SELinux from denying access.

8.3.4. Permissive Domains

When SELinux is running in permissive mode, SELinux does not deny access, but denials are logged for actions that would have been denied if running in enforcing mode. Previously, it was not possible to make a single domain permissive (remember: processes run in domains). In certain situations, this led to making the whole system permissive to troubleshoot issues.

Permissive domains allow an administrator to configure a single process (domain) to run permissive, rather than making the whole system permissive. SELinux checks are still performed for permissive domains; however, the kernel allows access and reports an AVC denial for situations where SELinux would have denied access.

Permissive domains have the following uses:

- They can be used for making a single process (domain) run permissive to troubleshoot an issue without putting the entire system at risk by making it permissive.
- They allow an administrator to create policies for new applications. Previously, it was

recommended that a minimal policy be created, and then the entire machine put into permissive mode, so that the application could run, but SELinux denials still logged. **audit2allow** could then be used to help write the policy. This put the whole system at risk. With permissive domains, only the domain in the new policy can be marked permissive, without putting the whole system at risk.

8.3.4.1. Making a Domain Permissive

To make a domain permissive, run the **semanage permissive -a *domain*** command, where *domain* is the domain you want to make permissive. For example, run the following command as the Linux root user to make the **httpd_t** domain (the domain the Apache HTTP Server runs in) permissive:

```
~]# semanage permissive -a httpd_t
```

To view a list of domains you have made permissive, run the **semodule -l | grep permissive** command as the Linux root user. For example:

```
~]# semodule -l | grep permissive
permissive_httpd_t 1.0
permissivedomains 1.0.0
```

If you no longer want a domain to be permissive, run the **semanage permissive -d *domain*** command as the Linux root user. For example:

```
~]# semanage permissive -d httpd_t
```

8.3.4.2. Denials for Permissive Domains

The **SYSCALL** message is different for permissive domains. The following is an example AVC denial (and the associated system call) from the Apache HTTP Server:

```
type=AVC msg=audit(1226882736.442:86): avc:  denied  { getattr } for
pid=2427 comm="httpd" path="/var/www/html/file1" dev=dm-0 ino=284133
scontext=unconfined_u:system_r:httpd_t:s0
tcontext=unconfined_u:object_r:samba_share_t:s0 tclass=file

type=SYSCALL msg=audit(1226882736.442:86): arch=40000003 syscall=196
success=no exit=-13 a0=b9a1e198 a1=bfc2921c a2=54dff4 a3=2008171 items=0
ppid=2425 pid=2427 auid=502 uid=48 gid=48 euid=48 suid=48 fsuid=48
egid=48 sgid=48 fsgid=48 tty=(none) ses=4 comm="httpd"
exe="/usr/sbin/httpd" subj=unconfined_u:system_r:httpd_t:s0 key=(null)
```

By default, the **httpd_t** domain is not permissive, and as such, the action is denied, and the **SYSCALL** message contains **success=no**. The following is an example AVC denial for the same situation, except the **semanage permissive -a httpd_t** command has been run to make the **httpd_t** domain permissive:

```
type=AVC msg=audit(1226882925.714:136): avc:  denied  { read } for
pid=2512 comm="httpd" name="file1" dev=dm-0 ino=284133
scontext=unconfined_u:system_r:httpd_t:s0
tcontext=unconfined_u:object_r:samba_share_t:s0 tclass=file

type=SYSCALL msg=audit(1226882925.714:136): arch=40000003 syscall=5
```

```
success=yes exit=11 a0=b962a1e8 a1=8000 a2=0 a3=8000 items=0 ppid=2511
pid=2512 auid=502 uid=48 gid=48 euid=48 suid=48 fsuid=48 egid=48 sgid=48
fsgid=48 tty=(none) ses=4 comm="httpd" exe="/usr/sbin/httpd"
subj=unconfined_u:system_r:httpd_t:s0 key=(null)
```

In this case, although an AVC denial was logged, access was not denied, as shown by **success=yes** in the **SYSCALL** message.

Refer to Dan Walsh's "Permissive Domains" blog entry for further information about permissive domains.

8.3.5. Searching For and Viewing Denials

This section assumes the *setroubleshoot*, *setroubleshoot-server*, *dbus* and *audit* packages are installed, and that the **auditd**, **rsyslogd**, and **setroubleshootd** daemons are running. Refer to Section 5.2, “Which Log File is Used” for information about starting these daemons. A number of tools are available for searching for and viewing SELinux denials, such as **ausearch**, **aureport**, and **sealert**.

ausearch

The *audit* package provides the **ausearch** utility. From the ausearch(8) manual page: "**ausearch** is a tool that can query the audit daemon logs based for events based on different search criteria" [14]. The **ausearch** utility accesses **/var/log/audit/audit.log**, and as such, must be run as the Linux root user:

Searching For	Command
all denials	**ausearch -m avc**
denials for that today	**ausearch -m avc -ts today**
denials from the last 10 minutes	**ausearch -m avc -ts recent**

To search for SELinux denials for a particular service, use the **-c *comm-name*** option, where *comm-name* "is the executable's name" [15], for example, **httpd** for the Apache HTTP Server, and **smbd** for Samba:

```
~]# ausearch -m avc -c httpd
```

```
~]# ausearch -m avc -c smbd
```

With each **ausearch** command, it is advised to use either the **--interpret** (**-i**) option for easier readability, or the **--raw** (**-r**) option for script processing. Refer to the ausearch(8) manual page for further **ausearch** options.

aureport

The *audit* package provides the **aureport** utility. From the aureport(8) manual page: "**aureport** is a tool that produces summary reports of the audit system logs" [16]. The **aureport** utility accesses **/var/log/audit/audit.log**, and as such, must be run as the Linux root user. To view a list of SELinux denials and how often each one occurred, run the **aureport -a** command. The following is example output that includes two denials:

```
~]# aureport -a
```

```
AVC Report
==============================================================
# date time comm subj syscall class permission obj event
==============================================================
1. 05/01/2009 21:41:39 httpd unconfined_u:system_r:httpd_t:s0 195 file
getattr system_u:object_r:samba_share_t:s0 denied 2
2. 05/03/2009 22:00:25 vsftpd unconfined_u:system_r:ftpd_t:s0 5 file read
unconfined_u:object_r:cifs_t:s0 denied 4
```

Refer to the aureport(8) manual page for further **aureport** options.

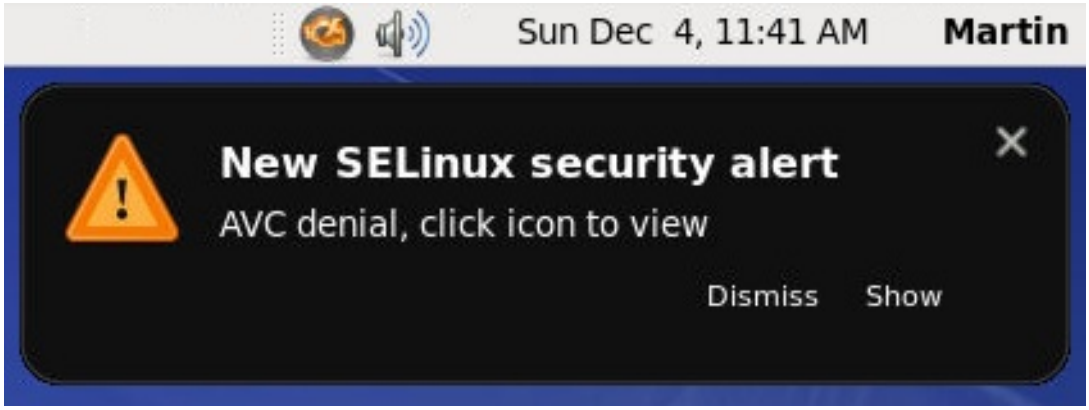
sealert

The *setroubleshoot-server* package provides the **sealert** utility, which reads denial messages translated by *setroubleshoot-server*. Denials are assigned IDs, as seen in **/var/log/messages**. The following is an example denial from **messages**:

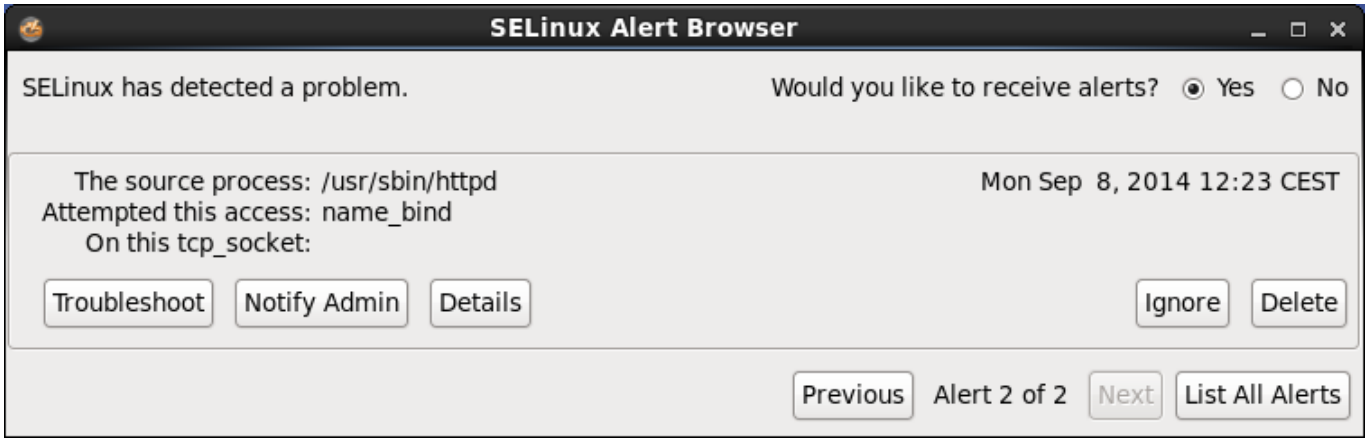
```
setroubleshoot: SELinux is preventing /usr/sbin/httpd from name_bind
access on the tcp_socket. For complete SELinux messages. run sealert -l
8c123656-5dda-4e5d-8791-9e3bd03786b7
```

In this example, the denial ID is **8c123656-5dda-4e5d-8791-9e3bd03786b7**. The **-l** option takes an ID as an argument. Running the **sealert -l 8c123656-5dda-4e5d-8791-9e3bd03786b7** command presents a detailed analysis of why SELinux denied access, and a possible solution for allowing access.

If you are running the X Window System, have the *setroubleshoot* and *setroubleshoot-server* packages installed, and the **setroubleshootd**, **dbus** and **auditd** daemons are running, a warning is displayed when access is denied by SELinux:



Clicking on **Show** launches the **sealert** GUI, which allows you to troubleshoot the problem:



Alternatively, run the **sealert -b** command to launch the **sealert** GUI. To view a detailed analysis of all denial messages, run the **sealert -l *** command.

See the sealert(8) manual page for further **sealert** options.

8.3.6. Raw Audit Messages

Raw audit messages are logged to **/var/log/audit/audit.log**. The following is an example AVC denial (and the associated system call) that occurred when the Apache HTTP Server (running in the **httpd_t** domain) attempted to access the **/var/www/html/file1** file (labeled with the **samba_share_t** type):

```
type=AVC msg=audit(1226874073.147:96): avc:  denied  { getattr } for
pid=2465 comm="httpd" path="/var/www/html/file1" dev=dm-0 ino=284133
scontext=unconfined_u:system_r:httpd_t:s0
tcontext=unconfined_u:object_r:samba_share_t:s0 tclass=file

type=SYSCALL msg=audit(1226874073.147:96): arch=40000003 syscall=196
success=no exit=-13 a0=b98df198 a1=bfec85dc a2=54dff4 a3=2008171 items=0
ppid=2463 pid=2465 auid=502 uid=48 gid=48 euid=48 suid=48 fsuid=48
egid=48 sgid=48 fsgid=48 tty=(none) ses=6 comm="httpd"
exe="/usr/sbin/httpd" subj=unconfined_u:system_r:httpd_t:s0 key=(null)
```

{ getattr }

The item in the curly brackets indicates the permission that was denied. The **getattr** entry indicates the source process was trying to read the target file's status information. This occurs before reading files. This action is denied due to the file being accessed having a wrong label. Commonly seen permissions include **getattr**, **read**, and **write**.

comm="*httpd*"

The executable that launched the process. The full path of the executable is found in the **exe=** section of the system call (**SYSCALL**) message, which in this case, is **exe="/usr/sbin/httpd"**.

path="*/var/www/html/file1*"

The path to the object (target) the process attempted to access.

scontext="*unconfined_u:system_r:httpd_t:s0*"

The SELinux context of the process that attempted the denied action. In this case, it is the SELinux context of the Apache HTTP Server, which is running in the **httpd_t** domain.

tcontext="*unconfined_u:object_r:samba_share_t:s0*"

The SELinux context of the object (target) the process attempted to access. In this case, it is the SELinux context of **file1**. Note that the **samba_share_t** type is not accessible to processes running in the **httpd_t** domain.

In certain situations, the **tcontext** may match the **scontext**, for example, when a process attempts to execute a system service that will change characteristics of that running process, such as the user ID. Also, the **tcontext** may match the **scontext** when a process tries to use more resources (such as memory) than normal limits allow, resulting in a security check to see if that process is allowed to break those limits.

From the system call (**SYSCALL**) message, two items are of interest:

- **success=*no***: indicates whether the denial (AVC) was enforced or not. **success=no** indicates the system call was not successful (SELinux denied access). **success=yes** indicates the system call was successful. This can be seen for permissive domains or unconfined domains, such as **initrc_t** and **kernel_t**.
- **exe="*/usr/sbin/httpd*"**: the full path to the executable that launched the process, which in this case, is **exe="/usr/sbin/httpd"**.

An incorrect file type is a common cause for SELinux denying access. To start troubleshooting, compare the source context (**scontext**) with the target context (**tcontext**). Should the process (**scontext**) be accessing such an object (**tcontext**)? For example, the Apache HTTP Server (**httpd_t**) should only be accessing types specified in the httpd_selinux(8) manual page, such as **httpd_sys_content_t**, **public_content_t**, and so on, unless configured otherwise.

8.3.7. sealert Messages

Denials are assigned IDs, as seen in **/var/log/messages**. The following is an example AVC denial (logged to **messages**) that occurred when the Apache HTTP Server (running in the **httpd_t** domain) attempted to access the **/var/www/html/file1** file (labeled with the **samba_share_t** type):

```
hostname setroubleshoot: SELinux is preventing httpd (httpd_t) "getattr"
to /var/www/html/file1 (samba_share_t). For complete SELinux messages.
run sealert -l 84e0b04d-d0ad-4347-8317-22e74f6cd020
```

As suggested, run the **sealert -l 84e0b04d-d0ad-4347-8317-22e74f6cd020** command to view the complete message. This command only works on the local machine, and presents the same information as the **sealert** GUI:

```
~]$ sealert -l 84e0b04d-d0ad-4347-8317-22e74f6cd020

Summary:

SELinux is preventing httpd (httpd_t) "getattr" to /var/www/html/file1
(samba_share_t).

Detailed Description:

SELinux denied access to /var/www/html/file1 requested by httpd.
/var/www/html/file1 has a context used for sharing by different program.
If you
would like to share /var/www/html/file1 from httpd also, you need to
```

```
change its
file context to public_content_t. If you did not intend to this access,
this
could signal a intrusion attempt.

Allowing Access:

You can alter the file context by executing chcon -t public_content_t
'/var/www/html/file1'

Fix Command:

chcon -t public_content_t '/var/www/html/file1'

Additional Information:

Source Context                unconfined_u:system_r:httpd_t:s0
Target Context                unconfined_u:object_r:samba_share_t:s0
Target Objects                /var/www/html/file1 [ file ]
Source                        httpd
Source Path                   /usr/sbin/httpd
Port                          <Unknown>
Host                          hostname
Source RPM Packages           httpd-2.2.10-2
Target RPM Packages
Policy RPM                    selinux-policy-3.5.13-11.fc12
Selinux Enabled               True
Policy Type                   targeted
MLS Enabled                   True
Enforcing Mode                Enforcing
Plugin Name                   public_content
Host Name                     hostname
Platform                      Linux hostname 2.6.27.4-68.fc12.i686 #1
SMP Thu Oct
30 00:49:42 EDT 2008 i686 i686
Alert Count                   4
First Seen                    Wed Nov  5 18:53:05 2008
Last Seen                     Wed Nov  5 01:22:58 2008
Local ID                      84e0b04d-d0ad-4347-8317-22e74f6cd020
Line Numbers

Raw Audit Messages

node=hostname type=AVC msg=audit(1225812178.788:101): avc:  denied  {
getattr } for  pid=2441 comm="httpd" path="/var/www/html/file1" dev=dm-0
ino=284916 scontext=unconfined_u:system_r:httpd_t:s0
tcontext=unconfined_u:object_r:samba_share_t:s0 tclass=file

node=hostname type=SYSCALL msg=audit(1225812178.788:101): arch=40000003
syscall=196 success=no exit=-13 a0=b8e97188 a1=bf87aaac a2=54dff4
a3=2008171 items=0 ppid=2439 pid=2441 auid=502 uid=48 gid=48 euid=48
suid=48 fsuid=48 egid=48 sgid=48 fsgid=48 tty=(none) ses=3 comm="httpd"
exe="/usr/sbin/httpd" subj=unconfined_u:system_r:httpd_t:s0 key=(null)
```

Summary

A brief summary of the denied action. This is the same as the denial in **/var/log/messages**. In this example, the **httpd** process was denied access to a file (**file1**), which is labeled with the **samba_share_t** type.

Detailed Description

A more verbose description. In this example, **file1** is labeled with the **samba_share_t** type. This type is used for files and directories that you want to export via Samba. The description suggests changing the type to a type that can be accessed by the Apache HTTP Server and Samba, if such access is desired.

Allowing Access

A suggestion for how to allow access. This may be relabeling files, enabling a Boolean, or making a local policy module. In this case, the suggestion is to label the file with a type accessible to both the Apache HTTP Server and Samba.

Fix Command

A suggested command to allow access and resolve the denial. In this example, it gives the command to change the **file1** type to **public_content_t**, which is accessible to the Apache HTTP Server and Samba.

Additional Information

Information that is useful in bug reports, such as the policy package name and version (**selinux-policy-3.5.13-11.fc12**), but may not help towards solving why the denial occurred.

Raw Audit Messages

The raw audit messages from **/var/log/audit/audit.log** that are associated with the denial. Refer to Section 8.3.6, “Raw Audit Messages” for information about each item in the AVC denial.

8.3.8. Allowing Access: audit2allow

Do not use the example in this section in production. It is used only to demonstrate the use of the **audit2allow** utility.

From the audit2allow(1) manual page: "**audit2allow** – generate SELinux policy allow rules from logs of denied operations" [17]. After analyzing denials as per Section 8.3.7, “sealert Messages”, and if no label changes or Booleans allowed access, use **audit2allow** to create a local policy module. After access is denied by SELinux, running the **audit2allow** command presents Type Enforcement rules that allow the previously denied access.

The following example demonstrates using **audit2allow** to create a policy module:

1. A denial and the associated system call are logged to **/var/log/audit/audit.log**:

```
type=AVC msg=audit(1226270358.848:238): avc:  denied  { write } for
pid=13349 comm="certwatch" name="cache" dev=dm-0 ino=218171
scontext=system_u:system_r:certwatch_t:s0
tcontext=system_u:object_r:var_t:s0 tclass=dir

type=SYSCALL msg=audit(1226270358.848:238): arch=40000003
syscall=39 success=no exit=-13 a0=39a2bf a1=3ff a2=3a0354 a3=94703c8
```

```
items=0 ppid=13344 pid=13349 auid=4294967295 uid=0 gid=0 euid=0
suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=(none) ses=4294967295
comm="certwatch" exe="/usr/bin/certwatch"
subj=system_u:system_r:certwatch_t:s0 key=(null)
```

In this example, **certwatch** (**comm="certwatch"**) was denied write access (**{ write }**) to a directory labeled with the **var_t** type (**tcontext=system_u:object_r:var_t:s0**). Analyze the denial as per Section 8.3.7, “sealert Messages”. If no label changes or Booleans allowed access, use **audit2allow** to create a local policy module.

2. With a denial logged, such as the **certwatch** denial in step 1, run the **audit2allow -w -a** command to produce a human-readable description of why access was denied. The **-a** option causes all audit logs to be read. The **-w** option produces the human-readable description. The **audit2allow** utility accesses **/var/log/audit/audit.log**, and as such, must be run as the Linux root user:

```
~]# audit2allow -w -a
type=AVC msg=audit(1226270358.848:238): avc:  denied  { write } for
pid=13349 comm="certwatch" name="cache" dev=dm-0 ino=218171
scontext=system_u:system_r:certwatch_t:s0
tcontext=system_u:object_r:var_t:s0 tclass=dir
 Was caused by:
  Missing type enforcement (TE) allow rule.

 You can use audit2allow to generate a loadable module to allow
this access.
```

As shown, access was denied due to a missing Type Enforcement rule.

3. Run the **audit2allow -a** command to view the Type Enforcement rule that allows the denied access:

```
~]# audit2allow -a


#============= certwatch_t ==============
allow certwatch_t var_t:dir write;
```

Important

Missing Type Enforcement rules are usually caused by bugs in SELinux policy, and should be reported in Red Hat Bugzilla. For Red Hat Enterprise Linux, create bugs against the **Red Hat Enterprise Linux** product, and select the **selinux-policy** component. Include the output of the **audit2allow -w -a** and **audit2allow -a** commands in such bug reports.

4. To use the rule displayed by **audit2allow -a**, run the **audit2allow -a -M *mycertwatch*** command as the Linux root user to create custom module. The **-M** option creates a Type Enforcement file (**.te**) with the name specified with **-M**, in your current working directory:

```
~]# audit2allow -a -M mycertwatch
```

```
******************** IMPORTANT ***********************
To make this policy package active, execute:

semodule -i mycertwatch.pp

~]# ls
mycertwatch.pp  mycertwatch.te
```

Also, **audit2allow** compiles the Type Enforcement rule into a policy package (**.pp**). To install the module, run the **semodule -i *mycertwatch.pp*** command as the Linux root user.

Important

Modules created with **audit2allow** may allow more access than required. It is recommended that policy created with **audit2allow** be posted to an SELinux list, such as fedora-selinux-list, for review. If you believe their is a bug in policy, create a bug in Red Hat Bugzilla.

If you have multiple denials from multiple processes, but only want to create a custom policy for a single process, use the **grep** command to narrow down the input for **audit2allow**. The following example demonstrates using **grep** to only send denials related to **certwatch** through **audit2allow**:

```
~]# grep certwatch /var/log/audit/audit.log | audit2allow -M
mycertwatch2
******************** IMPORTANT ***********************
To make this policy package active, execute:

~]# semodule -i mycertwatch2.pp
```

Refer to Dan Walsh's "Using audit2allow to build policy modules. Revisited." blog entry for further information about using **audit2allow** to build policy modules.

---

[12] Files in **/etc/selinux/targeted/contexts/files/** define contexts for files and directories. Files in this directory are read by the **restorecon** and **setfiles** commands to restore files and directories to their default contexts.

[13] The **semanage port -a** command adds an entry to the **/etc/selinux/targeted/modules/active/ports.local** file. Note that by default, this file can only be viewed by the Linux root user.

[14] From the ausearch(8) manual page, as shipped with the *audit* package in Red Hat Enterprise Linux 6.

[15] From the ausearch(8) manual page, as shipped with the *audit* package in Red Hat Enterprise Linux 6.

[16] From the aureport(8) manual page, as shipped with the *audit* package in Red Hat Enterprise Linux 6.

[17] From the audit2allow(1) manual page, which is available when the *policycoreutils-sandbox* package in Red Hat Enterprise Linux 6 is installed.

Chapter 9. Further Information

9.1. Contributors

- Domingo Becker – Translation – Spanish
- Dominick Grift – Technical Editor
- Daniel Cabrera – Translation – Spanish
- Murray McAllister – Red Hat Engineering Content Services
- James Morris – Technical Editor
- Eric Paris – Technical Editor
- Scott Radvan – Red Hat Engineering Content Services
- Daniel Walsh – Red Hat Security Engineering
- Geert Warrink – Translation – Dutch

9.2. Other Resources

The National Security Agency (NSA)

From the NSA Contributors to SELinux page:

Researchers in NSA's National Information Assurance Research Laboratory (NIARL) designed and implemented flexible mandatory access controls in the major subsystems of the Linux kernel and implemented the new operating system components provided by the Flask architecture, namely the security server and the access vector cache. The NSA researchers reworked the LSM-based SELinux for inclusion in Linux 2.6. NSA has also led the development of similar controls for the X Window System (XACE/XSELinux) and for Xen (XSM/Flask).

- Main SELinux website: http://www.nsa.gov/research/selinux/index.shtml.
- SELinux documentation: http://www.nsa.gov/research/selinux/docs.shtml.
- SELinux background: http://www.nsa.gov/research/selinux/background.shtml.

Tresys Technology

Tresys Technology are the upstream for:

- SELinux userland libraries and tools.
- SELinux Reference Policy.

SELinux News

- News: http://selinuxnews.org/wp/.
- Planet SELinux (blogs): http://selinuxnews.org/planet/.

SELinux Project Wiki

- Main page: http://selinuxproject.org/page/Main_Page.
- User resources, including links to documentation, mailing lists, websites, and tools: http://selinuxproject.org/page/User_Resources.

Fedora

- Main page: http://fedoraproject.org/wiki/SELinux.
- Troubleshooting: http://fedoraproject.org/wiki/SELinux/Troubleshooting.
- Fedora SELinux FAQ: http://docs.fedoraproject.org/.
- SELinux Managing Confined Services Guide: http://docs.fedoraproject.org/

The UnOfficial SELinux FAQ

http://www.crypt.gen.nz/selinux/faq.html

IRC

On Freenode:

- #selinux
- #fedora-selinux
- #security

Revision History

Revision 6-0	**Fri Oct 10 2014**	**Barbora Ančincová**
Red Hat Enterprise Linux 6.6 GA release of the book		
Revision 4-0	**Feb Fri 22 2013**	**Tomáš Čapek**
Release of the SELinux Guide for Red Hat Enterprise Linux 6.4		
Revision 3-0	**Wed Jun 20 2012**	**Martin Prpič**
Release of the SELinux Guide for Red Hat Enterprise Linux 6.3		
Revision 2-0	**Tue Dec 6 2011**	**Martin Prpič**
Release of the SELinux Guide for Red Hat Enterprise Linux 6.2		
Revision 1.9-0	**Wed Mar 3 2010**	**Scott Radvan**
Revision for Red Hat Enterprise Linux 6		